Der letzte Unix-Kurs vor der AutoBAhn

Skript zum

Online-Unix-Kurs

© 2005 Kerstin Rossek und Silke Wacker

Inhaltsverzeichnis

1 Unix / Linux für Anfänger **1**

2 Einordnung **2**
2.1 Begriffsklärung . . . 2
2.1.1 Die Geschichte von Linux . . . 3
2.1.2 GPL & Co. Ist Linux frei? . . . 8
2.1.3 Distributionen . . . 10
2.2 Abgrenzung zu anderen Betriebssystemen . . . 17
2.3 Aufgaben und Aufbau eines Betriebssystems . . . 19
2.3.1 Aufgaben eines Betriebssystems . . . 19
2.3.2 Aufbau eines Betriebssystems . . . 20

3 Hilfe unter Linux **23**
3.1 Man Pages . . . 23
3.1.1 Allgemeines . . . 23
3.1.2 Aufbau einer Man Page . . . 24
3.1.3 Sektionen der Man Pages . . . 24
3.1.4 X Man Pages . . . 26
3.2 Hilfesuche . . . 26
3.2.1 Die Befehle whatis und apropos . . . 27
3.2.2 Der info Befehl . . . 27
3.3 HOWTOs . . . 28

4 Nutzerverwaltung **32**
4.1 Multi User System . . . 32
4.2 Benutzer . . . 33
4.2.1 Eigenschaften eines Benutzers . . . 33
4.2.2 Benutzertypen . . . 34
4.2.3 Benutzer verwalten . . . 35

4.3 Benutzergruppen . . . 37
4.3.1 Benutzergruppen verwalten . . . 37
4.4 Benutzerklassen / Berechtigungsklassen . . . 38
4.4.1 Berechtigungsklassen . . . 39
4.5 Benutzerdateien . . . 40
4.5.1 Die Datei passwd . . . 40
4.5.2 Die Datei shadow . . . 40
4.5.3 Die Datei groups . . . 41

5 Dateiverwaltung 42
5.1 Dateisystemhierarchie . . . 42
5.1.1 Hierarchischer Aufbau . . . 42
5.1.2 File Hierarchie Standard (FHS) . . . 45
5.2 Dateisysteme . . . 46
5.2.1 Betriebsebenen auf der Dienste-Schicht . . . 46
5.2.2 Dateisysteme . . . 47
5.2.3 Inodes . . . 48
5.3 Befehle im Zusammenhang mit Dateien und Verzeichnissen . . . 48
5.4 Verweise: Symbolische- und Hard-Links . . . 50
5.4.1 Symbolischer Link . . . 50
5.4.2 Hard-Link . . . 50
5.4.3 Was ist nun besser? . . . 51
5.5 Zugriffsrechte . . . 52
5.5.1 Linux Access Control . . . 52
5.5.2 Setzen und Ändern von Zugriffsrechten . . . 53
5.6 Einbinden von Dateisystemen (Mounten) . . . 55
5.6.1 Mounten . . . 55
5.6.2 fstab und mtab . . . 55
5.6.3 fstab Dateisystemtypen . . . 56
5.7 Partitionierung . . . 57
5.7.1 Organisation . . . 57
5.7.2 Befehle . . . 59
5.8 Dateisystem-Integrität . . . 64
5.8.1 df - disk free . . . 64
5.8.2 du - disk usage . . . 65
5.8.3 fsck - file system check . . . 65

6 Programme/Prozesse **67**
6.1 Programme = Prozesse ? . . . 67
6.1.1 Programme und Prozesse . . . 67
6.1.2 Prozesse . . . 68
6.2 Prozesszustände . . . 69
6.2.1 Multitasking . . . 69
6.2.2 Prozesszustände . . . 70
6.2.3 Schnittstellen von Prozessen . . . 71
6.3 Streams, Umleitungen, Pipes . . . 71
6.3.1 Unix Streams . . . 71
6.3.2 Redirection - Umleitung . . . 73
6.3.3 Pipes . . . 73
6.4 Prozesskontrolle . . . 74
6.4.1 Überwachung von Prozessen . . . 74
6.4.2 Beeinflussung . . . 77
6.5 Daemons . . . 79
6.5.1 inetd . . . 79
6.6 init.d . . . 80
6.6.1 intit . . . 80
6.6.2 Runlevels . . . 80
6.6.3 Struktur von rc.d . . . 81

7 Shell **85**
7.1 Kommandozeile . . . 85
7.1.1 Bash Basics . . . 85
7.1.2 Kommandozeileneditor . . . 86
7.1.3 Automatische Kommando- und Dateinamenerweiterung (Tabulator-Taste) 86
7.1.4 History - das Gedächtnis der Kommandozeile . . . 86
7.1.5 Reservierte Worte und Zeichen mit besonderer Bedeutung . . . 87
7.1.6 Quotierung . . . 87
7.1.7 Erweiterungen . . . 88
7.1.8 Wichtige Dateien für die Bash . . . 89
7.2 Kommandos . . . 89
7.2.1 Wichtige Shell Kommandos . . . 89
7.2.2 Exit-Status . . . 90
7.2.3 Mehrere Kommandos in einer Kommandozeile . . . 90
7.2.4 Arithmetische Ausdrücke Berechnen . . . 91
7.2.5 Kommandosubstitition . . . 91

7.2.6 Aliase (Synonyme) . . . 92
7.3 Variablen . . . 92
7.3.1 Variablen - Setzen und Lesen . . . 92
7.3.2 Auswahl wichtiger Umgebungsvariablen . . . 93
7.4 Shellprogrammierung . . . 93
7.4.1 Wiederkehrende Tätigkeiten . . . 93
7.4.2 Kommentare . . . 94
7.4.3 Schleifen . . . 94
7.4.4 Verzweigungen . . . 96
7.4.5 Funktionen . . . 97

8 Der Editor vi **98**
8.1 Allgemeines zum Editor vi . . . 98
8.2 Modi & Kommandos . . . 100
8.2.1 Die drei Modi des Editor vi . . . 100
8.2.2 Wichtige Befehle im Kommandomodus . . . 102
8.2.3 Wichtige Befehle im Zeilenmodus . . . 102

9 Externe Kommandos **104**
9.1 Compilieren eines Quelltextes . . . 105
9.1.1 Das Kommando gcc . . . 105
9.1.2 Sourcedatei kompilieren . . . 105
9.2 grep . . . 107
9.3 Reguläre Ausdrücke . . . 108
9.3.1 Beispiele . . . 110
9.4 gzip und gunzip . . . 111
9.4.1 gzip . . . 111
9.4.2 gunzip . . . 112
9.4.3 Alternativen . . . 112
9.5 tar . . . 113
9.5.1 Beispiele . . . 113
9.5.2 Funktionsoptionen . . . 113
9.5.3 Weitere nützliche Optionen . . . 114
9.6 mtools . . . 114
9.7 find . . . 115
9.7.1 Nützliche Paramter . . . 115
9.7.2 Aktionen . . . 116
9.7.3 Beispiele . . . 116

9.8 Weitere Kommandos 116
9.9 Weitere Beispiele 117

10 Dienste **118**
10.1 telnet 118
10.1.1 Kommunikation via Telnet 119
10.1.2 Verwendung des Kommandos telnet 120
10.1.3 Nachteile von Telnet 120
10.2 ftp 121
10.2.1 Kommunikation via FTP 121
10.2.2 Verwendung des Kommandos ftp 121
10.2.3 FTP-Server mit einem Internetbrowser nutzen 123
10.2.4 FTP-Clients 123
10.2.5 Anonyme FTP-Server 124
10.3 ssh 124
10.3.1 Kommunikation mit ssh 124
10.3.2 Verbindung zu einem SSH-Server herstellen 125
10.3.3 SSH unter Windows 127
10.4 scp 127
10.4.1 Beispiele 128
10.4.2 Nützliche Paramter 128
10.4.3 SCP unter Windows 129

11 X Window System **130**
11.1 Allgemeines zum X Window System 130
11.2 XFree86 132
11.2.1 Konfiguration 133
11.2.2 Display-Manager 136
11.3 X.Org 136
11.4 Window Manager 137
11.5 Desktop-Systeme 137
11.5.1 KDE (The K Desktop Environment) 139
11.5.2 Gnome 142

A Literaturverzeichnis **I**

Abbildungsverzeichnis

2.1 Wettstein'sches Schichtenmodell [23] . . . 21
2.2 Die Schichten eines UNIX-Systems [20] . . . 22

3.1 Man Page zum Befehl du . . . 29
3.2 Man Page zum Befehl du unter xman . . . 30
3.3 info Dokumentation zum Befehl du . . . 31

4.1 Beispiel mehrere Prozesse mit unterschiedlichen Nutzerkennungen. (Kommando: ps aux) . . . 33

5.1 Verzeichnisbaum eines Linuxsystems [20] . . . 44
5.2 Sonderzeichen [20] . . . 45
5.3 Mode-Bits einer Datei . . . 54
5.4 Partitionen einer Festplatte (/dev/hda) . . . 59
5.5 Partitionieren mit fdisk - Hilfetext . . . 60
5.6 fdisk: Anzeige der Partitionen . . . 61
5.7 fdisk: Partitionstypen . . . 61
5.8 fdisk: neue Partition anlegen . . . 62
5.9 fdisk: neue Partitionstabelle . . . 63
5.10 mkfs: formatieren . . . 64

6.1 Elementare Prozesszustände [20] . . . 70
6.2 Standard-I/O eines Programms [20] . . . 72
6.3 Ausgabe des Befehls *ps* zur Anzeige aller laufenden Prozesse des Users . . . 75
6.4 Ausgabe des Befehls *pstree* zur Anzeige aller laufenden Prozesse des Users in Baumstruktur . . . 83
6.5 Ausgabe des Befehls *top* zur Anzeige aller laufenden Prozesse (Ausschnitt) . . . 84

7.1 Kommandozeileneditor Tasten [20] . . . 86
7.2 Bash History [20] . . . 87
7.3 Bash Sonderzeichen [20] . . . 87

8.1 Startbildschirm des Editor vi 99
8.2 Anzeige „New File“ im Editor vi 100
8.3 Anzeige der Zeilen und Zeichen im Editor vi 101
8.4 Die Modi des Editor vi [20] 102
8.5 Befehle im Kommandomodus [20] 103
8.6 Befehle im Zeilenmodus [20] 103

9.1 Konsolenausgabe des Kommandos gcc -v 105
9.2 Beispieldatei *lugmu.txt* 110

10.1 Telnet im OSI Schichtenmodell [1] 119
10.2 Schematische Darstellung einer FTP-Verbindung [5] 121
10.3 Schematische Darstellung der Kommunikation [15] 125
10.4 Aufruf von ssh-keygen 127

11.1 X Window System, Schichten und Komponenten [20] 131
11.2 Beispiel XF86Config 135
11.3 Window Maker 138
11.4 KDE 3.1 140
11.5 Gnome 2 142

1 Unix / Linux für Anfänger

Für viele Neueinsteiger und Umsteiger von anderen Betriebssystemen sind Linux und Unix auch nach einigen Semestern ein Buch mit sieben Siegeln. Woran liegt das? Linux ist wesentlich mächtiger als andere Betriebssysteme, was es jedoch auch komplexer macht. Dieser **Online-Unix/ Linux-Kurs** soll Grundlagen legen und dem Benutzer die Angst nehmen, sich mit diesem Betriebssystem zu beschäftigen und sich auszuprobieren. In Fachkreisen lautet eine Regel, um vernünftig mit Linux arbeiten zu können: Vergessen Sie die grafische Oberfläche! Effiziente Lösungen finden Sie nur auf Kommandozeilen-(Shell-)Ebene. Das Arbeiten mit dieser Shell ist ein großes Thema dieser Studienarbeit. Ein Highlight ist die **simulierte Shell**. Hier können einige grundlegende Befehle ausprobiert werden.

Denn wenn Sie mit dieser nicht umgehen können, werden Sie die wirklich relevanten Informationen und Dateien nicht finden, sie werden nicht wissen, was sich alles in Ihrem Heimatverzeichnis befindet und wer es alles lesen und eventuell manipulieren kann. Sie werden nie die Mächtigkeit erkennen, wenn Sie immer nur mit Officepaketen arbeiten.

Diese Studienarbeit ist ein Einstieg in das Thema Unix/ Linux. Sie soll einem User die Angst nehmen und Ihnen zeigen, dass Sie sich etwas trauen können.

Wir wünschen Euch gutes Gelingen und viel Spaß beim Erlernen der verschiedenen Themengebiete. Der eingebaute Test, soll euer Wissen überprüfen und euch helfen, Defizite zu erkennen und auszumerzen. Das absolute Novum ist die simulierte Shell. Dort könnt ihr direkt einige Befehle austesten und euch ausprobieren. Gutes Gelingen und viel Erfolg beim Linux/ Unix - Studium.

Ihre Silke Wacker und Kerstin Rossek
Alcatel SEL AG

Vorarbeit:
Alexander Rasl, IBM Deutschland GmbH
Alexander Wettley, Agilisys
Falk Winthuis, IBM Deutschland GmbH

2 Einordnung

Das folgende Kapitel „Einordnung“ soll Ihnen ein Überblick über die Eigenschaften und Entstehungsgeschichte von Linux geben. Des Weiteren wird Linux gegen eine Auswahl aktueller Betriebssysteme abgegrenzt. Abschließend werden hier der Zweck und die Aufgaben eines Betriebssystems ausführlicher betrachtet.

Die folgenden Punkte sind Inhalt dieses Kapitels.

- 2.1 Was ist Linux? Geschichte, Lizenzen, Distributionen
- 2.2 Abgrenzung Betriebssysteme
- 2.3 Aufgaben und Aufbau eines Betriebssystems

2.1 Begriffsklärung

Es soll hier zunächst einmal geklärt werden, was man unter dem Begriff „Linux“ eigentlich versteht. Prinzipiell ist mit dem Begriff „Linux“ nur der **Kernel** (allg. Betriebssystemkern) gemeint, also der Teil des Systems, der z.B. die Ressourcen für die laufenden Programme und die Anwender verwaltet. Obwohl der Kernel ein essentieller Teil des Systems ist, wäre er ohne zusätzliche Software nutzlos. Um diesen Kernel zu nutzen, ist weitere (teilweise sehr systemnahe) Software notwendig, also das eigentliche Betriebssystem. Beide Teile zusammen bilden ein vollwertiges Betriebssystem. Derzeit wird in den meisten Fällen die **Software des GNU-Projektes** in Verbindung mit dem Linux-Kernel verwendet. Heute sind beide Komponenten (der Linux-Kernel und die GNU-Betriebssoftware) kaum noch voneinander zu trennen. Korrekterweise müsste man also eigentlich, wenn man das Betriebssystem meint, von „GNU/Linux“ sprechen, da hier immer der Kernel (Linux) und die Systemwerkzeuge (GNU) gemeinsam gemeint sind. Es hat sich allerdings heute eingebürgert, dass umgangssprachlich unter dem Begriff „Linux“ die Kombination aus Kernel und Systemwerkzeugen gemeint ist, und wird in diesem Sinne auch in diesem Kurs verwendet.

Der wichtigste Unterschied gegenüber herkömmlichen Unix-Systemen besteht darin, dass Linux zusammen mit dem vollständigen Quellcode frei kopiert werden darf (Details dazu finden Sie im

Abschnitt 2.1.2 GPL & Co). Einschränkungen in der Funktionalität gibt es dagegen kaum; Linux ist in vielerlei Beziehung besser als so manches teure Unix-System. Es unterstützt eine größere Palette von Hardware-Komponenten und enthält in vielen Bereichen effizienteren Code.

Wichtig für das Verständnis von Linux ist auch, dass es nicht das „eine Linux“ gibt. Grund dafür sind eine Vielzahl sogenannter **Distributoren** (Details dazu in Kapitel 2.1.3 Distributionen). Dabei handelt es sich um unabhängige Organisationen, die zu den oben bereits genannten Komponenten von Linux zusätzliche Software sammeln, dies aufeinander abstimmen und dann vertreiben.

2.1.1 Die Geschichte von Linux

Nachdem Sie jetzt einen kleinen Einblick bekommen haben, was sich hinter dem Begriff „Linux“ verbirgt, soll in diesem Abschnitt der Ursprung von Linux beleuchtet werden. Die Geschichte von Linux beginnt eigentlich im Jahre 1991. Um aber die Entstehungsgeschichte von Linux ganz zu verstehen, muss man sich in das Jahr 1969 begeben. Dieses Jahr war der Startschuss für das Betriebssystem Unix, dem Linux, wie bereits angedeutet, ziemlich ähnlich ist.

„Das Betriebsystem UNIX“

Unix entsprang ironischerweise einem gescheiterten Projekt: Zu Beginn des Jahres 1969 gab es ein Gemeinschaftsprojekt des Massachusetts Institute of Technology (MIT), General Electric und den Bell Labs von AT&T, dass Ideen für eine neue Generation von Betriebssystemen gesammelt hatte und daran ging diese Ideen unter dem Namen Multics umzusetzen. Da weder Zeitplan noch Budget eingehalten werden konnten, zog sich Bell Labs sehr schnell aus dem Projekt zurück.

Ken Thompson und Denis Ritchie, zwei Mitarbeiter von Bell Labs, die an Multics mitgearbeitet hatten, waren von den Einfällen und Erfahrungen, die sie mit Multics gesammelt hatten, so beeindruckt, dass sie kurzerhand eine abgespeckte Version des ursprünglichen Multics selbst schrieben und unter dem Namen Unics, später UNIX, in die Welt setzten. Witzigerweise war eine Hauptmotivationen für UNIX jedoch ein Spiel, das Thompson in seiner Freizeit geschrieben hatte. UNIX gewann sehr schnell eine große Verbreitung innerhalb der Bell Labs. AT&T war es aufgrund einer kartellgerichtlichen Entscheidung verwehrt beliebig im kommerziellen Feld tätig zu werden, so auch im Falle von UNIX. Stattdessen lizensierte AT&T UNIX gegen nominelle Gebühren an Universitäten, an denen UNIX seinen ersten Siegeszug antrat.

Eine besondere Rolle kam dabei der University of California in Berkeley zu, die einen eigenen Zweig des UNIX - Systems hervorgebracht hat, die Berkeley Software Distribution (BSD). Sie war eng mit dem Quellcode von AT&T verwoben, zur Verwendung benötigte man also ebenfalls eine Lizenz

von AT&T.

UNIX zeichnete sich durch Eigenschaften aus, die heute selbstverständlich sind, aber zu dieser Zeit recht neu waren:

Ein hierarchisches Dateisystem: Die ersten DOS-Versionen von Mircosoft aus den 80er Jahren halten dies noch nicht.

Multitasking: Bei den Betriebssystemen Microsoft Windows und Apple MacOS war man erst in der zweiten Hälfte der 90er Jahre auf dem Weg dorthin.

Multi-User-Fähigkeit: Selbst die aktuelle Version von Mircosoft Windows (XP) kann zwar mehrere Benutzer verwalten (wie auch der Vorgänger Windows 2000), aber es kann zu jedem Zeitpunkt maximal ein Benutzer am System angemeldet sein und mit dem Betriebssystem arbeiten.

Netzwerkfähigkeit: Schon sehr früh wurden die UNIX-Kernel mit einem TCP/IP-Stack ausgestattet und bildeten schnell das Rückgrat des damals noch jungen Internets.

Plattformunabhängigkeit: Die meisten Betriebssysteme damals (und auch noch heute) waren auf einen bestimmten Prozessortyp zugeschnitten, und diese Abhängigkeit setzte sich in den Programmen fort.

Automatisierung: Zahlreiche Aufgaben lassen sich unter UNIX sehr einfach automatisieren.

1984 trennte sich AT&T von etlichen Tochterfirmen, womit auch ihr gestattet wurde, sich als gewöhnlicher Wettbewerber auf dem Computermarkt zu betätigen. Damit wurden auch Lizenzgebühren für UNIX drastisch angehoben und der Zugang zum Quellcode mehr und mehr eingeschränkt. Die Folge war, dass die Kooperation zwischen den Firmen, die UNIX kommerziell vermarkteten, immer mehr zurückging und jeder in „seiner“ UNIX-Version seine eigenen Erweiterungen und Verbesserungen einbaute, bis UNIX heillos in unterschiedliche Versionen aufgesplittert war. **SunOS** von Sun, **HP-UX** von Hewlett-Packard; **AIX** von IBM, **Ultrix** von Digital, **Sinix** von Siemens, auch Mircosoft versuchte sich auf dem UNIX-Markt mit **Xenix**. Ein großer Vorteil, die leichte Portierbarkeit der UNIX-Programme, drohte mit dieser Zersplitterung zu verschwinden und viele Stimmen prophezeiten auch ein mittelfristiges Ende von UNIX.

Um dieser Entwicklung entgegenzuwirken, wurden von verschiedenen Seiten Standards für UNIX spezifiziert. AT&T versuchte 1984, nach der Wandlung zu einem gewöhnlichen Wettbewerber, einen Standard zu schaffen: System V (V entspricht römisch 5, also „System Five“ gesprochen).

1985 brachte AT&T die „System V Interface Definition“ heraus. Dieses Dokument stellte ein Standard für die UNIX-Schnittstellen dar. Zusätzlich enthielt es auch eine Menge Werkzeuge, die

ein System auf die Konformität mit dem Standard V überprüfte. Diese von AT&T im Jahre 1983 freigegebene Version „UNIX System V“ war zu dieser Zeit die dominierende Version. Sie stellte den Versuch seitens AT&T dar, die Hersteller auf einen Standard zu einen. Wegen des Widerstandes, der unter anderem dadurch entstand, dass man sich nicht von einer einzigen Firma abhängig machen wollte, entstanden im Laufe der Zeit andere Standards, so z.B. POSIX (Portable Operating System based on UNIX). Weil AT&T alle Rechte an dem Namen UNIX hatte, wurde vom Institute of Electrical and Electronic Engineers (IEEE) dieser Name für diesen Standard gewählt. Ein anderes Beispiel hierfür ist **X/Open**: Das X/Open Konsortium ist ein Zusammenschluss verschiedener Computerhersteller, die einen De-Facto-Standard schaffen wollten. **1988** wurde der X/Open Portability Guide veröffentlicht. All diesen Standardisierungsversuchen blieb der Durchbruch verwehrt.

Erfolg hatte dagegen das GNU-Projekt, welches am Anfang der 80er Jahre am MIT begonnen wurde. Sein Ziel war es, von Grund auf ein neues, UNIX-ähnliches Betriebssystem zu schreiben, das frei verfügbar sein sollte. Bis Ende der 90er Jahre entstand eine beachtliche und leistungsstarke Sammlung an UNIX-Werkzeugen. Auch wenn das System bislang nicht vollständig ist, konnten sich die GNU-Werkzeuge dennoch auf vielen UNIX-Varianten etablieren. So wurden die GNU-Werkzeuge ein systemübergreifender Quasi-Standard. Die Entwicklungsmethode des GNU-Projektes hatte erreicht, woran die proprietären Standardisierungsversuche bislang gescheitert waren.

Auch von akademischer Seite wurde der immer zugeknöpfteren Haltung der UNIX-Vertreiber begegnet. Wie bereits zu Anfang beschrieben, wurde der Quellcode von UNIX durch AT&T gegen nominelle Gebühren den Universitäten zur Verfügung gestellt. Dieser Code wurde vielerorts in Tutorials als Beispiel für die Arbeitsweise eines Betriebssystems verwendet. Nachdem AT&T jedoch den Quellcode unter Verschluss brachte, fiel diese Möglichkeit weg. Andrew S. Tanenbaum, Informatik-Professor an der Freien Universität Amsterdam, entschloss sich daher, für seine Studenten eine eigene Version von UNIX zu schreiben, die nichts mit dem urheberrechtlich geschützten Code von AT&T zu tun hatte. Nach zwei Jahren harter Arbeit brachte er sein System unter dem Namen Minix heraus. Es war weniger für die praktische Arbeit, sondern in erster Linie als Lehrobjekt gedacht. Dennoch wurde es von vielen Studenten auch praktisch auf dem heimischen PC eingesetzt, da es im Gegensatz zu den kommerziellen UNIX - Varianten für einen moderaten Preis zu haben war. Allerdings stieß Minix in diesem Einsatzgebiet sehr schnell an seine Grenzen. Viele seiner Anwender machten Tanenbaum Vorschläge und schickten Patches für Erweiterungen und Verbesserungen. Tanenbaum allerdings war damit sehr zurückhaltend. Da er Minix in erster Linie als Tutorial sah, kam es ihm mehr auf eine knappe und klare Struktur als auf eine möglichst umfassende Funktionalität an. Ein Minix-Anwender mit Namen **Linus Torvalds** gab sich damit nicht zufrieden. Das GNU-System war bis auf den Kernel vollständig, aber das Release des GNU-Kernels mit Namen HURD schien noch auf sich warten zu lassen. Um die zeitliche Lücke bis dahin zu füllen, begann er selbst einen Kernel zu schreiben, der sehr rasch unter dem Namen **Linux** Verbreitung fand und eine große Entwickler- und Benutzergemeinde zusammenbrachte. Da die

meisten Entwickler auf UNIX-Varianten arbeiteten, auf denen die GNU-Werkzeuge liefen, lag es nahe, den Linux-Kernel so einzurichten, dass er zusammen mit den GNU-Werkzeugen verwendet werden konnte: **GNU/Linux**.

Zur gleichen Zeit löste sich BSD aus seiner ursprünglichen Abhängigkeit von AT&T: Eine Gruppe von BSD-Entwicklern ersetzte alle Anweisungen im Quellcode, die noch AT&T beigesteuert waren, durch neue und erstritt in einem langwierigen Gerichtsverfahren für BSD die Freiheit. Daraus gingen die Projekte FreeBSD, NetBSD und OpenBSD hervor, die auch eine beträchtliche Verbreitung gefunden haben und manchmal als „Linux-Vettern“ bezeichnet werden.

Wie die Geschichte zeigt, gibt es nicht das eine Betriebssystem UNIX. Vielmehr handelt es sich um eine „Familie“ von Betriebssystemen.

Linux in Jahreszahlen

Seither hat sich Linux zu einem bedeutenden UNIX entwickelt; Kommerzielle UNIX-Anbieter haben Marktanteile an Linux verloren und mussten neue Strategien entwickeln. Nicht selten mündeteten diese Überlegungen in offener Unterstützung für Linux, dessen weitere Verbreitung ohnehin nicht mehr zu stoppen war.

Im Folgenden nun ein Abriss wichtiger Jahreszahlen in der Erfolgsgeschichte von Linux:

01.01.1991: Der 21-jährige finnische Student **Linus Benedict Torvalds** beginnt, aufbauend auf dem Minix Betriebssystem, ein unixartiges Betriebssystem für AT-386-Computer zu schreiben.

01.02.1992: Linus Torvalds verteilt die **Version 0.12** seines Kernels per anonymous FTP im Internet, was zu einem sprunghaften Anstieg der Zahl interessierter Benutzer führt. Da diese Zahl so groß wird, dass die nötige Kommunikation nicht mehr per Email zu bewältigen ist, wird in den Usenet News die Gruppe alt.os.linux ins Leben gerufen. Dies hat zur Folge, dass eine explosionsartige Weiterentwicklung des Systems im ganzen Internet entsteht und von Linus Torvalds fortan koordiniert wird.

01.03.1993: Bereits über 100 Programmierer arbeiten am Linux-Code mit. Durch **Anpassung des Linux-Kernels an die GNU-Umgebung** der Free Software Foundation (FSF) im Jahre 1993 wachsen Möglichkeiten von Linux erneut stark an, da man nun auf eine große Sammlung an vorhandener freier Software und Tools zurückgreifen kann, die unter Linux laufen.

01.04.1994: Mit der **Linux-Version 1.0** wird der Betriebssystem-Kernel **netzwerkfähig** und die Anzahl der Linuxnutzer steigt auf 100.000 an. Ein wichtiger Schritt, der ebenfalls im Jahre

1994 geschieht, ist auch die Anpassung einer **grafischen Benutzerschnittstelle** (GUI) auf Linux. Diese wird von einer weiteren „Non-Profit-Gruppe“ dem XFree86-Projekt, beigesteuert. Linus Torvalds stellt auch den Quelltext des Linux-Kernels offiziell unter die GNU General Public License (mehr dazu im folgenden Abschnitt 2.1.2). Somit ist die freie Existenz von Linux gesichert.

01.05.1995: Linux wird **auf die Plattformen Digital (DEC) und Sun Sparc portiert**. Damit kann sich das neue Betriebssystem nun mit vollem Schwung auf den vielen Plattformen ausbreiten.

01.06.1996: Mit der neuen **Version 2.0** des Linux-Kernels können nun **mehrere Prozesse gleichzeitig** angesteuert werden. Linux verliert langsam seinen Bastlerstatus und wird zu einer ernst zu nehmenden Alternative für Firmen.

01.07.1997: Nun erscheinen wöchentlich neue, aktualisierte Versionen des Linux-Kernels. Verschiedene namenhafte Firmen beginnen ihre Software auf Linux portieren: Netscape seinen Webbrowser, Applixware seine Office-Anwendung und die Software AG ihre Datenbank Adabas D. Damit gibt es immer mehr kommerzielle Software-Pakete für Linux.

01.08.1998: Das Desktop-Projekt KDE wird gestartet. Es arbeiten etwa 750 Programmierer am dieser heute am weitesten verbreiteten Desktopumgebung. Viele **namenhafte Hardware- und Softwarehersteller kündigen die Portierung ihrer Produkte auf Linux an**. Darunter finden sich Firmen wie IBM und Compaq (fusionierte im Jahr 2002 mit Hewlett Packard), die Linux als Betriebssysteme auf ihren Computern unterstützen. Informix (übernommen durch IBM im Jahr 2001) und Oracle entwicklen ihre Datenbanken fortan auch unter Linux. Netscape gibt die Quellen seines Webbrowsers frei und lässt die zukünftige Entwicklung durch das Mozilla-Projekt vorantreiben.

01.09.1999: Die **Kernelversion 2.2** erscheint. Sie verfügt über einen verbesserten SMP-Support und einen überarbeiteten Netzwerkcode. Ein neues Desktop-Projekt mit dem Namen GNOME wird begonnen. IBM kündigt die Portierung von Lotus Domino Notes an und propagiert seine Linux-Strategie.

01.10.2000: KDE 2.0 erscheint. IBM kündigt für 2001 Investitionen in Linux in der Höhe von einer Milliarde US-Dollar an. Sun veröffentlicht den Quellcode von StarOffice unter der LGPL (Lesser GPL) und legt damit den Grundstein für OpenOffice.

01.11.2001: Die **Kernelversion 2.4** erscheint. Der Kernel **unterstützt bis zu 64 Gbyte Arbeitsspeicher und 64Bit-Dateisysteme**. Ebenso werden die USB-Schnittstelle und das Journaling Filesystem (JFS) unterstützt. Linux ist portiert auf den IBM Großrechner iSeries (ehemals AS/400).

01.12.2002: Das OpenOffice-Projekt bringt OpenOffice in der Version 1.0 auf den Markt. Es ist ein komplettes Office-Paket mit Textverarbeitung, Tabellenkalkulation, Präsentationsmodul und läuft nicht nur unter Linux. Der Open Source Browser Mozilla wird nach vier Jahren in der Version 1.0 veröffentlicht. Auch bei den Desktops wird die nächste Runde eingeläutet: KDE 3.0 erscheint im Frühling, GNOME 2.0 zur Jahresmitte.

13.01.2003: Linus Torvalds wechselt von seinem bisherigen Arbeitgeber Transmeta in das Open Source Development Lab (OSDL). Dort wird er in Zukunft auch beruflich seine Zeit der Arbeit am Linux-Kernel widmen. Linux findet zusehends Verbreitung auf Embedded Systemen. Der **Münchener Stadtrat** hat sich am 28.5. auf Grund einer Studie für die **Umstellung seiner 14.000 Computer von Microsoft Windows auf Linux** entschieden. KDE Desktop 3.1 erscheint. OpenOffice wird in der Version 1.1 veröffentlicht, welches etliche Erweiterungen gegenüber den Vorgängerversionen bietet. Samba erscheint in der Version 3.0, welche gerade im Einsatzbereich als Domain-Controller viele Erweiterungen und Verbesserungen erfahren hat. Auch eine Integration in das von Windows 2000 eingeführte „Active Directory“ ist nun möglich. GNOME Desktop 2.4 erscheint. Die Entwicklerserie 2.5 des Linux-Kernels wird geschlossen und in die Serie 2.6.0-test übergeführt. Am 17. Dezember wird Version 2.6.0 des Linux-Kernels freigegeben.

Ende 2004: Kernel-Version 2.6.10

2.1.2 GPL & Co. Ist Linux frei?

Linux ist „frei“ - aber was bedeutet das eigentlich? Oft wird frei mit kostenlos verwechselt. Es stimmt zwar, das Linux auch kostenlos verfügbar ist (zumindest über das Internet); der Begriff „frei“ bezieht sich aber auch und vor allem auf die Verfügbarkeit des gesamten Quellcodes (Open Source). Damit sind gewisse Komplikationen verbunden: Was passiert, wenn eine Firma den Linux-Code verwendet, in einigen Punkten erweitert und das System anschließend verkauft? Auch das ist erlaubt, allerdings mit einer Einschränkung; Der Programmcode des neuen Systems muss abermals frei verfügbar sein. Diese Regelung stellt sicher, dass Erweiterungen am System allen Anwendern zugute kommen. Ziel der Entwickler von GNU und Linux war es also, ein System zu schaffen, dessen Quellen frei verfügbar sind und es auch bleiben. Um einen Missbrauch auszuschließen, ist Software, die im Sinne von GNU entwickelt wurde und wird, durch die GNU General Public License (GPL) geschützt. Hinter der GPL steht die Free Software Foundation (FSF). Diese Organisation wurde gegründet, um qualitativ hochwertige Software frei verfügbar zu machen.

GNU General Public License (GPL)

Die Kernaussage der **GPL** besteht darin, dass zwar jeder den Code verändern und sogar die resultierenden Programme verkaufen darf, dass aber gleichzeitig der Anwender beziehungsweise Käufer das Recht auf den vollständigen Code hat und diesen ebenfalls verändern und wieder kostenlos weitergeben darf. Jedes GNU-Programm muss zusammen mit der vollständigen GPL weitergegeben werden. Durch die GPL geschützte Software ist also nicht mit Public-Domain-Software zu verwechseln, die vollkommen ungeschützt ist. Die GPL schließt damit aus, dass jemand ein GPL-Programm weiterentwickeln und verkaufen kann, ohne die Veränderungen öffentlich verfügbar zu machen. Jede Weiterentwicklung ist somit ein Gewinn für alle Anwender. Die deutsche Übersetzung finden Sie unter `http://www.gnu.de/gpl-ger.html`.

GNU Libary General Public License (LGPL)

Neben der GPL existiert noch die Variante **LGPL** (Libary GPL). Der wesentliche Unterschied zur GPL besteht darin, dass eine derart geschützte Bibliothek auch von kommerziellen Produkten genutzt werden darf, deren Code nicht frei verfügbar ist. Ohne die LGPL könnten GPL-Bibliotheken nur wieder für GPL-Programme genutzt werden, was in vielen Fällen eine unerwünschte Einschränkung für kommerzielle Programmierer wäre. Die LGPL wurde durch ihren Nachfolger, der GNU Lesser General Public License, zu Beginn des Jahres 1999 abgelöst.

Andere Lizenzmodelle im Linux-Umfeld

Durchaus nicht alle Teile einer Linux-Distribution unterliegen den gleichen Copyright-Bedingungen! Während der Kernel und viele Tools der GPL unterliegen, gelten für manche Komponenten und Programme andere rechtliche Bedingungen. Im folgenden sind beispielhaft vier Fälle angeführt:

- Beispielsweise gilt für das **X Window System** eine eigene Lizenz, da dieses ursprünglich vom MIT entwickelt wurde. Die jetzige Lizenz ist von einer früheren Lizenz des MIT abgeleitet.

- Für manche **Netzwerk**-Tools gilt die BSD-Lizenz. Dabei handelt es sich um das Lizenzmodell, welches für die bereits erwähnten Betriebssysteme OpenBSD, NetBSD und FreeBSD verwendet wird. Dieses Modell ist insofern liberaler als die GPL, als die kommerzielle Nutzung ohne die Freigabe des Codes zulässig ist. Die Lizenz ist daher vor allem für kommerzielle Programmierer interessant, die Produkte entwickeln möchten, deren Code sie nicht veröffentlichen wollen.

- Für einige Programme gelten **Doppellizenzen**. Beispielsweise können Sie den Datenbank-Server MySQL für OpenSource-Projekte beziehungsweise für die innerbetriebliche Anwendung gemäß der GPL kostenlos einsetzen. Wenn Sie hingegen ein kommerzielles Produkt auf der Basis von MySQL entwickeln und mit MySQL verkaufen möchten (ohne Ihren Quellcode zur Verfügung zu stellen), kommt die kommerzielle Lizenz zum Einsatz. Das bedeutet, dass die Weitergabe von MySQL in diesen Fall kostenpflichtig wird.

- Andere Programme sind **dediziert kommerziell**, obwohl auch in solchen Fällen die kostenlose Nutzung meist zulässig ist. Ein bekanntes Beispiel ist der Adobe Acrobat Reader zum Lesen von Dokumenten im PDF Format: Zwar ist das Programm unter Linux kostenlos erhältlich (und darf auch in Firmen kostenlos eingesetzt werden), aber der Quellcode zu diesem Programm ist nicht erhältlich.

Manche Distributionen kennzeichnen die Produkte, bei denen die Nutzung oder Weiterentwicklung eventuell lizenzrechtliche Probleme verursachen könnte. So befinden sich bei SuSE Linux alle derartigen Programm-Pakete in der Serie pay oder werden nicht mitgeliefer (z.B. Grafikkartentreiber von NVidia). Bei Debian Linux werden solche Pakete gleich gar nicht mitgeliefert oder sie befinden sich im Verzeichnis „non-free“.

Im Allgemeinen können Sie davon ausgehen dass Sie alle Programme, die Sie mit einer Linux-Distribution erhalten haben, auch kostenlos nutzen dürfen. Es ist aber nicht immer so, dass Sie davon abgeleitete eigene Produkte ohne weiteres weiterverkaufen dürfen. Wenn Sie Software-Entwickler sind, müssen Sie sich in die bisweilen sehr verwirrende Problematik der unterschiedlichen Software-Lizenzen einarbeiten.

2.1.3 Distributionen

Als Linux-Distribution wird eine Einheit bezeichnet, die aus dem eigentlichen Betriebssystem (Kernel und systemnahe Software) und seinen Zusatzprogrammen besteht. Eine Distribution ermöglicht eine einigermaßen rasche und bequeme Installation unter Linux. Alle Distributionen werden in der Form von CD-ROMs oder auch DVD-ROMs verkauft. Die meisten Distributionen sind darüber hinaus auch im Internet verfügbar. Wegen der riesigen Datenmengen (Hunderte von Megabytes bis hin zu mehreren Gigabytes) ist das Kopieren einer Distribution via Internet oder eine direkte Installation über das Netz aber zumeist nur in Universitäten bzw. Firmen mit Breitbandanbindung an das Internet möglich. Allerdings ist es auch möglich Zuhause per DSL in einer relativ kurzen Zeit (mehrere Stunden) eine Distribution übers Internet zu installieren.

Die Distributionen unterscheiden sich vor allem durch folgende Punkte voneinander:

- **Umfang, Aktualität:** Die Anzahl, Auswahl und Aktualität der mitgelieferten Programme und Bibliotheken unterscheidet sich von Distribution zu Distribution. Manche Distributionen überbieten sich in der Anzahl der mitgelieferten CD-ROMs. Um den Anwendern bei der Installation bzw. späteren Updates ein ständiges CD-Wechseln zu ersparen, werden manche Distributionen auch als DVD-ROMs ausgeliefert. Vorreiter war hier die Distribution der Firma SuSE.

- **Installations-und Konfigurationswerkzeuge:** Die mitgelieferten Programme zur Installation, Konfiguration und Wartung des Systems helfen dabei, Hunderte von Konfigurationsdateien des Systems auf einfache Weise richtig einzustellen. Funktionierende Installations- und Konfigurationstools stellen eine enorme Zeitersparnis dar.

- **Konfiguration des Desktops (z.B. KDE, Gnome):** Manche Distributionen lassen dem Anwender die Wahl zwischen KDE, Gnome und eventuell auch verschiedenen Window Managern. Andere legen den Anwender auf eines dieser Systeme fest. Es gibt aber auch Unterschiede in der Detailkonfiguration von KDE oder Gnome - zum Beispiel inwieweit deren Startmenüs mit den tatsächlich installierten Programmen übereinstimmen.

- **Hardware-Erkennung und -Konfiguration:** Linux kommt zwar nicht mit allen PC-Hardware-Komponenten zurecht, aber doch mit ziemlich vielen. Angenehm ist natürlich, wenn die Distribution die vorhandene Hardware automatisch erkennt und damit umgehen kann. Gelingt dies nicht, ist oft eine mühsame Konfiguration in Handarbeit erforderlich, die Linux-Einsteiger meist überfordert.

- **Landesspezifische Anpassung:** Manche Distributionen vermitteln noch immer den Eindruck, Englisch sei die einzige Sprache dieser Welt. Andere Distributionen sind speziell für den Einsatz in nicht englischsprachigen Ländern vorbereitet. Das betrifft nicht nur das Tastatur-Layout, sondern auch die verfügbaren Zeichensätze, die Sprache der Online- Dokumentation u.v.m. Ist man jedoch der englischen Sprache nicht abgeneigt, so gibt es Anpassungen für US- und UK-Englisch, ja sogar für Irisch-Englisch (Debian). In der Zwischenzeit gibt es sogar Übersetzungen für kleinere Sprachräume, z.B. KDE auf Plattdeutsch.

- **Paketsystem:** Die Verwaltung von Linux-Anwendungsprogrammen erfolgt durch Pakete (Packages) als modulare Einheiten. Das Paketsystem hat ein Einfluss darauf, wie einfach die Nachinstallation zusätzlicher Programme beziehungsweise die Aktualisierung (Update) vorhandener Programme ist. Zur Zeit sind drei zueinander mehr oder weniger inkompatible Paketsysteme üblich:

  - RPM (verwendet von Caldera, Mandrake, Red Hat, SuSE und TurboLinux)
  - DEB (verwendet von Debian, Corel, Progeny und Storm Linux)
  - TGZ (verwendet von Slackware)

Es gibt jedoch die Möglichkeit, z.B. RPM in Debian-Pakete umzuwandeln und vice-versa.

- **Wartung, Beseitigung von Sicherheitslücken:** Linux ist ein sich dynamisch veränderndes System. Oft gibt es nach der Fertigstellung einer Distribution noch wichtige Neuerungen; immer wieder werden Sicherheitsmängel in diversen mitgelieferten Programmen entdeckt. Eine gute Distribution zeichnet sich dadurch aus, dass sie (insbesondere bei Sicherheitsproblemen) sehr rasch Updates im Internet zur Verfügung stellt. Manche Distributionen versuchen mittlerweile einerseits den Vorgang von Sicherheits-Updates weitestgehend zu automatisieren, und daraus andererseits einen kostenpflichtige Zusatzleistung zu machen. Wie weit sich diese Idee durchsetzen wird, ist momentan noch nicht abzusehen.

- **Live System:** Einige wenige Distributionen ermöglichen den Betrieb von Linux direkt von einer CD-ROM oder DVD. Das ist zwar langsam und inflexibel, ermöglicht aber ein vergleichsweise einfaches Ausprobieren von Linux. Zudem stellt eine Live CD-ROM eine ideale Möglichkeit dar, ein auf der Festplatte vorhandenes aber defektes Linux-System zu reparieren.

- **Microsoft Windows-Installation:** Einige Distributionen ermöglichen die Installation von Linux in ein Verzeichnis bzw. in eine große Datei einer MS Windows-9x/SE/ME-Partition. Das ist langsam und hat zahllose Nachteile, erspart aber die Partitionierung der Fesplatte. Wie beim Live System ist auch diese Variante vor allem zum Ausprobieren interessant.

- **Hardware-Unterstützung:** Alle Distributionen für Intel-kompatible Prozessoren laufen auf jedem Standard-PC. Wenn Sie spezielle Hardware verwenden (Multiprozessor-Mainboards, RAID-Festplattensysteme, Notebooks etc.), hängt es aber stark von der Distribution ab, wie gut derartige Hardware von den Installations- und Konfigurations-Tools unterstützt wird. Hier sind große, weit verbreitete Distributionen meist im Vorteil. Von SuSE und RedHat gibt es Beispielsweise spezielle Enterprise-Distributionen welche auch Großrechner-Architekturen wie z.B. die „zSeries“ von IBM unterstützen.

- **Ziel-Plattform (CPU-Architektur):** Viele Distributionen sind nur für Intel-kompatible Prozessoren erhältlich. Es gibt aber auch Distributionen für andere Rechnerplattformen, zum Beispiel DEC Alpha, SUN Sparc und Macintosh PowerPC.

- **Dokumentation:** Manche Distributionen werden mit Handbüchern (in elektronischer oder gedruckter Form) ausgeliefert.

- **Support & Service:** Bei manchen Distributionen bekommen Sie kostenlos Hilfe bei der Installation (via Email oder Telefon). Das Ausmaß des gebotenen Supports schlägt sich meist sehr direkt auf den Preis nieder.

- **Mitgelieferte kommerzielle Software:** Bei manchen Distributionen werden nicht nur die frei verfügbaren Linux-Pakete mitgeliefert, sondern auch lizenzpflichtige Programme. Auch dies erhöht den Preis der Distributionen.
- **Lizenz:** Die meisten Distributionen sind uneigenschränkt kostenlos über das Internet erhältlich. Bei einigen Distributionen gibt es hier aber Einschränkungen. Beispielsweise stellen nicht alle Distributionen so genannte ISO-Images zur Verfügung, mit denen sich Anwender leicht selbst die Installations CD-ROMs brennen können (d.h. ohne die Distribution zu kaufen).

Manche Distributionen erlauben zwar die kostenlose Weitergabe, nicht aber den Weiterverkauf von CD-ROMs. (Da Linux und die meisten mitgelieferten Programme an sich frei erhältlich sind beschränkt sich das Verkaufsverbot meist auf die Installations- oder Konfigurationssoftware; bei SuSE galt dies beispielweise bis vor kurzem für das Programm YaST. Dieses steht alledings seit der Version SuSE Linux 9.1 unter der GPL und darf somit auch frei kopiert werden.) Weitergabeeinschränkungen gibt es auch, wenn mit der Distribution kommerzielle Software mitgeliefert wird. Am restriktivsten war Caldera: Dessen Linux-Distributionen dürfen nur auf einen einzigen Rechner installiert werden, was in der Linux-Welt vollkommen unüblich ist.

Die Behauptung Linux sei frei, steht scheinbar im krassen Widerspruch zu dem Preis, die für die meisten besseren Distributionen verlangt wird (oft 50 Euro und mehr). Der Grund ist aber leicht verständlich: Obwohl Linux und die meisten Anwendungsprogramme tatsächlich kostenlos über das Internet bezogen werden können, erfordert die Zusammenstellung einer aktuellen Distribution eine Menge Zeit und „Know-how“. Ein gutes Installationsprogramm allein (das auch programmiert und gewartet werden muss) ist den Preis einer Distribution oft schon wert! Es kann von allem Linux- bzw. Unix-Neulingen eine Menge Zeit bei der Installation und Konfiguration ersparen. Auch die Produktion von einer oder mehreren CD-ROMs, oft begleitet von einem Handbuch, kostet Geld. Nicht zu vernachlässigen ist schließlich das Angebot eines persönlichen Supports bei Installationsproblemen. Teuer wird eine Distribution auch dann, wenn kommerzielle Software mitgeliefert wird.

Die „richtige“ Linux-Distribution

Die Frage, welche Distribution die beste sei, welche wem zu empfehlen sei, artet leicht in einen Glaubenskrieg aus. Wer sich einmal für eine Distribution entschieden und sich an deren Eigenheiten gewöhnt hat, steigt nicht so schnell auf eine andere Distribution um. Ein Wechsel der Distribution ist nur durch eine Neuinstallation möglich, bereitet also einige Mühe. Kriterien für die Auswahl einer Distribution sind die Aktualität ihrer Komponenten (achten Sie auf die Versionsnummer des Kernels und wichtiger Programme, etwa des C-Compilers), die Qualität der Installations- und Konfigurationstools, der angebotene Support, mitgelieferte Handbücher etc. Dass die meisten

Linux-Distributionen wirklich uneingeschränkt frei verfügbar sind, erkennen Sie unter anderem daran, dass es von vielen kommerziellen Distributionen „Billig“ CD-ROMs mit den sogenannten FTP-Versionen, GPL-Versionen oder Download- Versionen gibt. Der Name dieser Version ergibt sich daraus, dass es sich um jene Dateien handelt, die kostenlos im Internet (meist auf einem FTP-Server) verfügbar sind. Diesen Versionen fehlen die kommerziellen Komponenten, die von der jeweiligen Distribution lizensiert wurden. Außerdem gibt es keine Dokumentation und keinen Support. Lassen Sie sich aber vom günstigen Preis der FTP-Versionen nicht blenden: Gerade für Einsteiger ist ein gutes Handbuch oder die Möglichkeit, während der ersten Monate eine Support-Abteilung kontaktieren zu können, sehr wertvoll.

Der folgende Überblick über die **wichtigsten verfügbaren Distributionen** (in alphabetischer Reihenfolge und ohne Anspruch auf Vollständigkeit!) soll eine erste Orientierungshilfe geben:

- **Caldera:** Caldera Open Linux war eine der ersten Linux-Distributionen, die sich explizit an kommerzielle Anwender wandte (Mittlerweile erheben alle Mitbewerber ebenfalls diesen Anspruch.). Die Distribution wurde jedoch im August 2002 in die Produktpalette vom UNIX-Anbieter SCO eingegliedert und besteht seitdem in dieser Form nicht mehr.

- **Corel:** Corel Linux wurde im Jahr 1999 sehr medienwirksam eingeführt und sollte eine sehr einfache und benutzerfreundliche Distribution werden. Obwohl dies ansatzweise sehr gut gelang, blieb Corel Linux der Erfolg verwehrt. Im September 2001 verkaufte Corel seine Linux-Abteilung an die Firma Xandros.

- **Debian:** Während sich hinter den meisten hier geannten Distributionen Firmen stehen, die mit ihren Distributionen Geld verdienen möchten, stellt Debian in dieser Beziehung eine Ausnahme dar: Die Distribution wird von engagierten Linux-Anwendern zusammengestellt, die größten Wert auf Stabilität und die Einhaltung der Spielregeln „freier“ Software legen. Manche Ideen dieser Distribution - etwa die professionelle Paketverwaltung - waren für andere Distributionen richtungsweisend und sind diesen in manchen Aspekten immer noch voraus. (Versuchen Sie einmal, ihre Distribution zu aktualisieren, ohne den Rechner neu zu starten!) Debian ist im Laufe der letzten Jahre zwar zunehmend benutzerfreundlicher geworden, für Linux-Einsteiger ist diese Distribution aber wegen des schwer zu bedienbaren Paketverwaltungsprogramms dselect nach wie vor ungeeignet (vgl. Progeny Debian).

- **Mandrake:** Mandrake Linux ist gewissermaßen eine benutzerfreundliche Variante zu Red Hat Linux. Die Distribution ist von Red Hat abgeleitet und insofern weitgehend kompatibel, als die meisten Software-Pakete untereinander austauschbar sind. Mandrake Linux zeichnet sich aber durch eigenständige und einfacher zu bedienende Installations- und Konfigurationsprogramme aus. Ein weiterer Vorteil besteht darin, dass mit Mandrake Linux in der Regel mehr

und aktuellere Softwarepakete mitgeliefert werden (manchmal auf Kosten der Stabilität). Hinter Mandrake steht die Firma MandrakeSoft mit Firmensitz in Frankreich.

- **Progeny Debian:** Wie der Name bereits andeutet, ist diese Distribution von Debian abgeleitet. Progeny Debian ist eine sehr neue Distribution (die erste Version wurde im Jahr 2001 ausgeliefert) und unterscheidet sich von Debian vor allem durch komfortablere Installations- und Konfigurationswerkzeuge. Da aber sämtliche Zusatzfunktionen von Progeny Debian in die Entwicklung von Debian Linux nach und nach eingeflossen sind, wurde die Entwicklung einer seperaten Distribution eingestellt.

- **Red Hat:** Die Red Hat Distribution ist einer der am besten gewarteten Linux-Distributionen, die zurzeit erhältlich sind. Die Distribution dominiert insbesondere den amerikanischen Markt. Die Paketverwaltung auf Basis des RPM-Formats (eine Eigenentwicklung von Red Hat) wurde mittlerweile von vielen anderen Distributionen übernommen. Neben der Red Hat Orginaldistribution gibt es eine ganze Reihe davon abgeleiteter Distributionen, die sich durch diverse Verbesserungen oder Sprachanpassungen (z.B. eine Version in spanischer Sprache) auszeichnen. Red Hat Linux ist insbesondere für die Verwendung als Server sehr beliebt, weil Sicherheits-Updates oft rascher verfügbar sind als bei anderen Distrbutionen. Auch legt die Firma Red Hat bei der Auswahl von Paketen und Versionen für eine Distribution ein größeres Augenmerk auf Stabilität (anstatt einfach die aktuellste verfügbare Versionen mitzuliefern). Im Vergleich zu anderen Distributionen sind die Konfigurationshilfen allerdings unübersichtlich organisiert und zum Teil schwierig zu bedienen. Red Hat ist also nicht unbedingt die optimale Distribution für Linux-Einsteiger.

  Die Bezeichnung Red Hat Linux beschreibt allerdings nur noch die kommerziell vertriebene Distribution. Die Fedora Community ist für das Fedora Core Linux zuständig, welches von der letzten frei verfügbaren Red Hat Distribution abgeleitet wurde.

- **Slackware:** Die Slackware war eine der ersten verfügbaren Linux-Distributionen. Bezüglich Wartung und Installationskomfort kann sie allerdings nicht mehr mit den anderen hier genannten Distributionen mithalten. Viele Slackware-Anwender bevorzugen ihre Distribution aber gerade deswegen, weil das Augenmerk eher auf Kontinuität und Stabilität denn auf schöne Installations- und Konfigurations-Tools gelegt wird.

- **Storm Linux:** Diese Distribution ist von Debian abgeleitet, bietet aber attraktivere Installations- und Konfigurationstools als diese. Jedoch ist seit April 2001 die Homepage vom Distributor Stormix Technologies inaktiv. Die Distribution ist offiziell nicht mehr verfügbar und es wird auch kein Support mehr gegeben.

- **SuSE:** SuSE-Linux ist dank der hohen Aktualität, der riesigen Anzahl vorkonfigurierter Pakete, den umfassenden Handbüchern und der hervorragenden Wartung die in Europa am

weitesten verbreitete Distribution. SuSE-Linux ist in zahlreichen Sprachen (Deutsch, Englisch etc.) erhältlich. SuSE ist eine sehr benutzerfreundliche Distribution. Das Administrationstool YaST hilft nicht nur bei vielen Konfigurationsproblemen, es löst auch ähnlich wie Debian automatisch eventuelle Abhängigkeitskonflikte zwischen Software-Paketen, die bei der Paketinstallation auftreten können. YaST steht seit der Version SuSE Linux 9.1 unter der GPL; dies war jedoch lange Zeit nicht so. Aus diesem Grund durfte man SuSE-Linux zwar im Freundeskreis frei kopieren, es war aber nicht zulässig, SuSE CD-ROMs billig zu verkaufen (wie dies bei vielen anderen Distributionen üblich ist). Zu den größten Nachteilen von SuSE zählte das Konfigurationskonzept (vgl. Datei */etc/rc.config*), das inkompatibel zu allen anderen Distributionen war und vor allem bei der Administration stört. Man findet es nur bei den älteren Linux-Distributionen, da dieses Konzept in den neueren Versionen ersetzt wurde (Die Abkürzung SuSE steht übrigens für Gesellschaft für Software und Systementwicklung).

- **Turbo Linux:** Diese von der Firma Pacific HighTech (PHT) zusammengestellte Distribution wurde speziell für die Verwendung in Asiatischen Raum optimiert und ist dort (nach eigenen Angaben) Marktführer. Die Distribution ist aber auch in einer englischen Version erhältlich.

- **Minimal-Distributionen:** Neben diesen großen Distributionen gibt es im Internet einige Zusammenstellungen von Miniatursystemen (bis hin zum kompletten Linux-System auf einer einzigen Diskette!). Diese Distributionen basieren zumeist auf alten (und daher kleineren) Kernel-Versionen. Sie sind vor allem für Spezialaufgaben konzipiert, etwa für Wartungsarbeiten oder um ein Linux-System ohne eigentliche Installation verwenden zu können (direkt von einer oder mehreren Disketten oder einer CD-ROM/ DVD-ROM). Das ist praktisch, wenn Sie Linux vorrübergehend auf einen fremden Rechner nutzen möchten, dessen Festplatte Sie nicht neu partitionieren wollen oder dürfen. Zwei gute Beispiele für Linux-Distributionen diesen Types ist das Linux Router Project (Auf einer einzigen Diskette sind alle Programme vorhanden, um einen alten 486er PC in einen Router für ein kleines Netzwerk umzuwandeln.) und KNOPPIX (Eine recht vollwertige Distribution, die sich von einer CD-ROM oder DVD-ROM starten läßt und das vorhandene System meist vollständig erkennt!).

Einen ziemlich guten Überblick über die momentan verfügbaren Linux-Distributionen (egal, ob kommerziellen oder anderen Ursprungs) finden Sie im Internet auf den folgenden Seiten:

- The Linux Information Headquarters

- LWN.net

Viele Distributionen, kein Standard !?!

Wie aus dem vorangegangenen Abschnitten ersichtlich ist, gibt es nicht „das Linux“. Man hat die Auswahl aus einer Vielzahl von Distributionen, die zwar gewisse Gemeinsamkeiten haben (z.B. Kernel, Paketverwaltung etc.), aber viele Unterschiede (Konfigurationstools, Softwareumfang etc.). Diese Tatsache kann sich als besonders lästig herausstellen bei der Installation von Programmen, die nicht mit der Distribution mitgeliefert werden (und insbesondere bei kommerziellen Programmen). Eine fehlende oder veraltete Programmbibliothek ist oft die Ursache dafür, dass ein Programm nicht läuft. Die Problembehebung ist insbesondere für Linux-Einsteiger fast unmöglich. Manche Firmen gehen inzwischen so weit, dass ihre Softwareprodukte nur eine ganz bestimmte Distribution unterstützen. Natürlich entscheidet sich jede Firma für eine andere Distribution.

Um diese Probleme zu beseitigen, wurde vor einigen Jahren das Linux Standard Base Project (LSB) ins Leben gerufen. Im Juli 2001 haben die Projektteilnehmer die LSB-Spezifikation 1.0 veröffentlicht. Es handelt sich dabei um Vorgaben, die alle LSB konformen Distributionen einhalten müssen. Dabei ist die Spezifikation in einem gemeinsamen Teil und architekturbezogenen Teil zu unterscheiden. Die aktuelle Spezifikation besteht in der Version 1.3. Die zurzeit aktuelle Version SuSE 9.1 Professional ist zum Beispiel für 32bit Intel-kompatible Architekturen mit dem 1.3 Standard konform. Ein anderer Schritt in Richtung Standardisierung von den unterschiedlichen Distributionen ist UNITEDLINUX. Dabei handelt es sich um eine Distribution, die von den Firmen Connectiva S.A., SCO Group, SuSE Linux AG und Turbolinux Inc. unterstützt wird. Im November 2002 wurde die Version 1.0 herausgegeben. Ziel ist es, einen weltweiten Linux-Standard zu schaffen.

Abschließend kann man sagen, dass zwar jede Distribution ihre Eigenheiten hat, aber dennoch die Gemeinsamkeiten überwiegen. Schließlich nutzen alle Distributionen zum Beispiel die zuvor genannnten GNU-Werkzeuge.

2.2 Abgrenzung zu anderen Betriebssystemen

In diesem Abschnitt sollen ein wenig die Unterschiede zwischen Linux und anderen Betriebssystemen aufgezeigt werden. Die Auflistung stellt lediglich eine Auswahl dar und hat keinen Anspruch auf Vollständigkeit, da das den Rahmen diese Kurses sprengen würde:

- **Apple MacOS:** Das Apple eigene Betriebssystem MacOS ist für die von Apple hergestellten Rechner vorgesehen. Der Grund liegt darin, dass diese Rechner keine Intel-kompatible Architektur haben, sondern es sich bei diesen um PowerPCs handelt. Es bietet eine enorm hohe Benutzerfreundlichkeit und wird bevorzugt für Multimedia-Anwendungen eingesetzt.

Bemerkenswert ist, dass in der neuesten Version des Betriebssystems (MacOS X; röm. 10) das gesamte Betriebssystem auf einen FreeBSD-Kernel portiert wurde.

- **Microsoft Windows XP:** Das aktuelle Desktop PC Betriebssystem von der Firma Mircosoft besitzt eine umfangreichere Unterstützung von verschiedensten Hardwarekomponenten im Vergleich zu seinem Vorgänger Windows 2000. Ebenfalls unterstützen die meisten Software-Hersteller, je nach Anwendungsgebiet der Software, primär das Microsoft Betriebssystem, da dieses die weiteste Verbreitung genießt. Des weiteren wurden die Benutzerfreundlichkeit und Multimedia-Funktionalität verbessert. Jedoch kann in dieser aktuellen Version von Microsoft Windows maximal ein Benutzer zu jedem Zeitpunkt angemeldet sein. Weiterhin ist zu bemerken, dass Windows XP, wie auch seine Vorgänger, Opfer vieler Viren- und Wurm-Attacken durch das Internet ist, was sich unter dem Aspekt der Lauffähigkeit und Stabilität nachteilig auswirkt.

- **Sun Solaris, IBM AIX, HP-UX:** Hierbei handelt es sich, wie bereits in der Geschichte von Linux eingeführt, um „Ableger“ des klassischen UNIX. Sie wurden im Laufe der Jahrzehnte durch die jeweiligen Firmen weiterentwickelt. Sie werden vornehmlich als Betriebssystem für die Großrechner der jeweiligen Firma eingesetzt.

- **zOS:** Das 64Bit-Betriebssystem für IBM zSeries-Mainframes ist ein typisches Großrechnerbetriebssystem. Das Betriebssystem ist sehr hardwarenah für eben die Mainframes von IBM entwickelt worden und nutzt deren Möglichkeiten vollständig aus. Im Vergleich zu PC-Desktop Betriebssystem, oder auch UNIX-Varianten für andere Großrechner, verfolgt zOS komplett andere Konzepte: Installation und Wartung von Programmen ist sehr sicher ausgelegt, ebenso sind wichtige (und auch viele weniger wichtige) Systemfunktionen redundant mit hohen Sicherheitsfunktionen ausgestattet.

- **zVM:** VM steht für „Virtuelle Maschine“ - mit diesem Großrechnerbetriebsystem von IBM kann man in virtuellen Umgebungen (ähnlich VMware) nahezu beliebige Betriebssysteme laufen lassen, so unter anderem auch Linux-Systeme. Diese müssen jedoch speziell für die Großrechnerarchitekturen (also z.B. zSeries) kompiliert sein, da Intel x86-Programme einen komplett anderen Binärcode haben.

- u.v.m.

Dieser Abschnitt macht deutlich, dass Linux zum jetztigen Zeitpunkt sicherlich nicht für alle Anwendungsgebiete geeignet ist, aber mehr und mehr an Attraktivität für die verschiedenen Einsatzgebiete von Betriebssystemen gewinnt.

2.3 Aufgaben und Aufbau eines Betriebssystems

Der folgende Abschnitt soll einen kleinen Überblick über die Aufgaben und den Aufbau eines Betriebssystems liefern. Ein umfassende und ausführliche Klassifizierung von Betriebssystemen würde sicherlich zum Einen den Rahmen dieses Kurses sprengen, zum Anderen das Ziel dieses Kurses verfehlen. Im Folgenden soll es Ihnen ermöglichen die noch folgenden Themengebiete in dem Gesamtkontext eines Betriebssystems, im Besonderen den von Linux, einzuordnen.

2.3.1 Aufgaben eines Betriebssystems

Ein Betriebssystem hat folgende grundlegende Aufgaben:

- **Verbergen der Komplexität der Maschine vor dem Anwender (Abstraktion):** Es gibt eine Vielzahl verschiedener Rechnerarchitekturen (Intel-kompatibel, PowerPC,...) und eine Vielzahl unterschiedlichster Hardwarekomponenten (Festplatten, Grafikkarten, Netzwerkkarten, Drucker,...). Beides erscheint in Variationen, dadurch daß es eine große Zahl an Herstellern für Rechnersysteme und Hardwarekomponenten gibt. All dies bleibt dem Benutzer des Betriebssystems verborgen.

- **Bereitstellen einer Benutzerschnittstelle:** Dem Benutzer muss eine Schnittstelle bereitgestellt werden, mit der mit dem Betriebssystem interagieren kann. Ein konkretes Beispiel wäre zum Beispiel eine grafische Oberfläche mit einen Navigationsmenü und grafischen Symbolen (vgl. KDE Desktopumgebung oder ein Kommandozeilen-Interpreter), in dem definierte Kommandos per Tastatur eingegeben werden (wie sie zum Beispiel ausschließlich beim „klassischen“ UNIX zur Verfügung stand).

- **Bereitstellen einer normierten Programmierschnittstelle:** Das Betriebssystem stellt eine Menge von Grundfunktionen bereit, mit denen programmiert werden kann, um das Betriebssystem um zusätzliche Funktionen zu erweitern. Auf diese Schnittstellen greifen zum Beispiel die Entwickler von Anwendungssoftware (z.B. ein Text Editor) zurück.

- **Verwaltung der Ressourcen der Maschine:** Mit Resourcen ist die gesamte Hardware gemeint, auf die das Betriebssystem aufsetzt und Zugriff hat, zum Beispiel Prozessor(-en), Hauptspeicher, Festplatten und andere Medien, Drucker, etc., aber auch logische Ressourcen wie zum Beispiel die Rechenzeit. Ein Beispiel wäre das Senden von Druckaufträgen an einen Drucker durch den Benutzer. Da der Drucker nur ein Dokument zu einem Zeitpunkt ausdrucken kann, regelt das Betriebssystem die Abarbeitung der Druckaufträge unabhängig von der Anzahl.

- **Verfolgung von Schutzstrategien bei dieser Ressourcenbereitstellung:** Am Beispiel der Ressource Arbeitsspeicher: auf einem Betriebssystem sind meistens mehrere Anwendungen installiert, die durchaus zum gleichen Zeitpunkt laufen können. Dabei benötigt jede Anwendung zur Laufzeit Speicherbereiche aus dem Arbeitsspeicher des Rechners, um zum Beispiele Werte temporär abzulegen (zum Beispiel eingelesene Eingaben von der Kommandozeile). Der Arbeitsspeicher wird jedoch nicht direkt durch die Anwendung allokiert, sondern die Anwendung fordert einen Speicherbereich benötigter Größe.

- **Koordination von Prozessen:** Ein Prozess ist ein Programm während der Ausführung. Ein Prozess kann sowohl aus Anwendungen resultieren, die durch den Benutzer gestartet worden sind, als auch durch systemnahe Programme, die schon beim Start des Betriebssystems initialisiert und gestartet werden. Im laufenden Betrieb des Betriebssystems sind meist Prozesse gleichzeitig aktiv (vgl. Multi-Tasking-Fähigkeit von UNIX). Das Betriebssystem koordiniert dabei die Interaktion zwischen den Prozessen. Ein Beispiel wäre das Warten eines Prozesses auf den Abschluss eines anderen Prozesses.

Die zuvor beschriebenen Aufgaben machen eines deutlich: die Aufgaben eines Betriebssystems haben unterschiedliche Granularität, das heißt einen unterschiedlich Grad der Abstraktion hinsichtlich des Rechnersystems und seiner Hardwarekomponenten. Der logische Schluss ist ein Schichtenmodell des Betriebssystems, in dem die unterschiedlichen Aufgaben erfüllt werden.

2.3.2 Aufbau eines Betriebssystems

Die im folgenden dargestellte Abbildung zeigt das Wettstein'sche Schichtenmodell nach Prof. em. Dr. Horst Wettstein. Von der untersten Schicht „Infrastruktur“ bis hin zur obersten Schicht „Steuerung“ steigt der Grad der Abstraktion zum eigentlichen Rechnersystem und dessen Hardwarekomponenten. Dabei handelt sich um ein sehr abstraktes Schichtenmodell, in dass sich eigentlich jedes bestehende Betriebssystem einteilen läßt (siehe Abbildung 2.1).

Die einzelnen Schichten sind wie folgt zu verstehen:

- **Infrastruktur:** Auf dieser untersten Schicht sind grundlegende Dinge wie Prozesse, Adreßräume (vgl. Arbeitsspeicher) und die Kommunikation zwischen Prozessen definiert. In dieser Schicht findet man auch die bereits oben genannte Programmierschnittstelle.

- **Dienste:** Mit der Dienstschicht werden Dinge wie zum Beispiel das Dateisystem, die Partitionierung der Festplatte, Treiber für Hardwarekomponenten, Bildschirmfenster (im Falle einer grafischen Benutzerschnittstelle) oder die Möglichkeit eine Datei anzulegen verbunden.

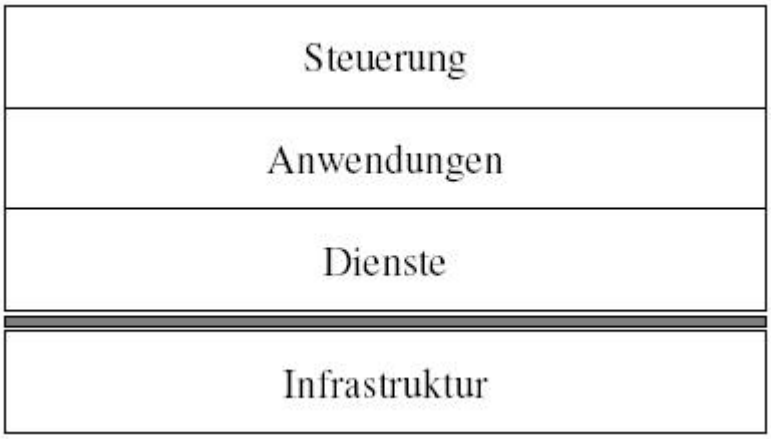

Abbildung 2.1: Wettstein'sches Schichtenmodell [23]

- **Anwendungen:** Diese Schicht ist die Nutz-Schicht mit den Anwendungen, die der Benutzer des Rechnersystems ausführen will. Ein anschauliches Beispiel wäre ein auf dem Betriebssystem installierter Text Editor.
- **Steuerung:** In dieser Schicht wird die Steuerung des Rechners durch den Benutzer ermöglicht, insbesondere das Starten von Anwendungen. Bei einem Betriebssystem mit einer grafischen Benutzerschnittstelle würde ein Klick mit der Maus durch diese Schicht ermöglicht.

Nach dieser genaueren Betrachtung des Wettstein'schen Schichtenmodells wird ein weiterer Aspekt deutlich: Jede übergeordnete Schicht greift auf Funktionen der Unterschicht zu.

Ein Beispiel:

- Der Benutzer startet ein Textverarbeitungsprogramm durch einen Klick mit der Maus auf das dazugehörige grafische Symbol innerhalb der grafischen Benutzerschnittstelle (Steuerung).
- Das installierte Textverarbeitungsprogramm wird gestartet (Anwendungen).
- Die grafische Oberfläche (Fenster, Schaltflächen, etc.) des Textverarbeitungsprogramms baut sich innerhalb der grafischen Benutzerschnittstelle auf (Dienste).
- Das Textverarbeitungsprogramm erhält einen eigenen Prozess und einen eigenen Adressraum im Arbeitsspeicher (Infrastruktur).

Die Abbildung 2.2 zeigt das konkrete Schichtenmodell eines UNIX-Systems.

Die neuen Begriffe, die Sie dieser Abbildung entnehmen können, werden in die den folgenden den Kapiteln näher erläutert.

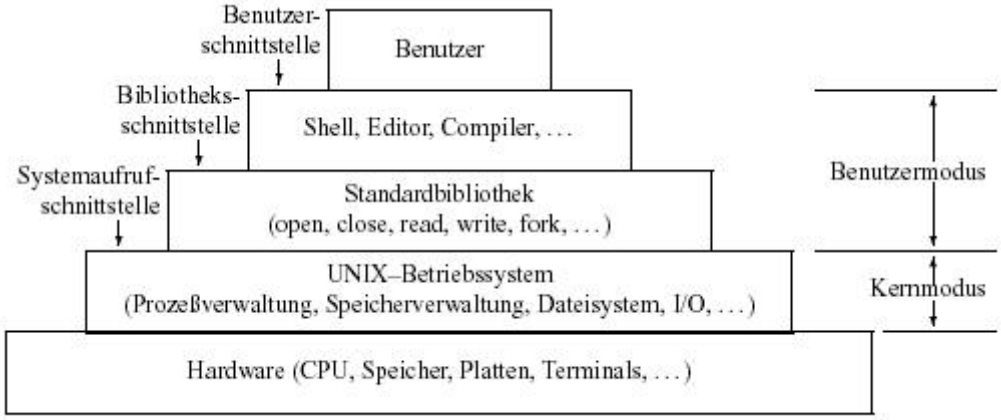

Abbildung 2.2: Die Schichten eines UNIX-Systems [20]

3 Hilfe unter Linux

Zu Linux gibt es unglaublich viel Dokumentationen, die teilweise gleich mitgeliefert werden und teilweise im Internet zu finden sind. Das Kapitel „Hilfe unter Linux“ stellt drei verschiedene Möglichkeiten vor, Hilfe zu verschiedenen Problem- bzw. Aufgabenstellungen unter Linux zu finden.

Die folgenden Punkte sind Inhalt dieses Kapitels.

- 3.1 Man-Pages
- 3.2 Hilfesuche
- 3.3 HOWTOs

3.1 Man Pages

3.1.1 Allgemeines

Die Man Pages („Manual-Seiten“) sind in Summe eine Art großes Benutzerhandbuch für Linux. Sie werden standardmäßig bei einer Installation einer Linux-Distribution auch mitinstalliert. Sie bieten den Vorteil, dass man Sie einfach nach einem Schlagwort durchsuchen kann. Auf diese Weise erhält man sofortigen Zugriff auf die Informationen, die man benötigt. Es entfällt das gewohnte langwierige Suchen wie man es von gedruckten Benutzerhandbüchern kennt. Es gibt eine Man Page für die meisten Kommandos, die auf dem System verfügbar sind (Es ist üblich, dass bei der Installation einer neuen Anwendung die passenden Man Pages ebenfalls dem System hinzugefügt werden), für wichtige Dateien (z.B. Konfigurationsdateien), Bibliotheksfunktionen, shells, Sprachen, Laufwerke und Geräte und andere Besonderheiten (z.B YaST2 bei SuSE Linux).

Die Man Pages sind generell für Diejenigen geschrieben, die schon eine Idee haben, in welchem Kontext ein gesuchter Begriff zu sehen ist, aber unabhängig davon, wieviel Erfahrung man im Umgang mit Linux hat. Wenn man zum Beispiel die Man Pages nach einem Kommandozeilen-Befehl durchsucht, sollte man grob eine Ahnung haben, welchem Zweck dieser Befehl dient. Die Man Page kann mit einem Tastendruck auf die „Q“ Taste verlassen werden.

Hilfe zu einem Kommando, etc. erhält man mit folgender Eingabe auf der Kommandozeile

`man` *befehl*

Witzigerweise gibt es für den man-Befehl ebenfalls eine Man Page (vgl. `man man`).

3.1.2 Aufbau einer Man Page

Der Aufbau jeder Man Page folgt einer definierten Struktur, die an dieser Stelle näher betrachtet werden soll. Dabei hängt der genaue Inhalt vom Begriff ab, für den man die Man Page abruft. Abbildung 3.1 zeigt einen Auszug aus der Man Page für den Befehl `du`, der Auskunft über den benötigten Speicherplatz von Dateien gibt. Der Aufbau der Man Pages für Befehle ist fast immer gleich.

Dem Kopf der Man Page kann man den Namen des Befehls („du“) entnehmen, den man gesucht hat und die Sektion (später mehr dazu), zu der die angezeigte Man Page gehört, hier die Sektion 1 „User Commands“. Die weitere Struktur lässt sich wie folgt aufgliedern:

- **NAME:** Benennung des Befehls mit kurzer Erläuterung, Einordnung.
- **ÜBERSICHT:** Übersicht der Befehlssyntax, d.h. der Anordnung der möglichen Parameter und diverser Optionen.
- **BESCHREIBUNG:** Anmerkungen und Beschreibungen zu den möglichen Optionen des Befehls.
- **AUTOR:** Angabe des Autors (der Autoren, des Projekts), der verantwortlich für die Entwicklung (und Pflege des Programms) verantwortlich ist.
- **FEHLERMELDUNG:** Angabe eines Kontaktes (z.B. Email-Adresse), an den man sich wenden sollte im Falle eines Programmfehlers.
- **URHEBERRECHT:** Erläuterungen zum lizenzrechtlichen Hintergrund des Programms.
- **SIEHE AUCH:** Verweise auf weitere Dokumentation oder anverwandte Programme.

3.1.3 Sektionen der Man Pages

Wie zuvor angedeutet, sind die Man Pages in Sektionen eingeteilt:

- **Kommandos (Sektion 1):** Die Benutzerprogramme, die man zum Beispiel von der Kommandozeile aufrufen kann. Diese Sektion wird am häufigsten benutzt, enthält die meisten Einträge und wächst mit der Anzahl der installierten Packages.

- **System-Aufrufe (Sektion 2):** System-Aufrufe werden meist von Kernel- oder Hardware-Programmierern benötigt. Für einfache Benutzer sind sie meist uninteressant. Sie heißen bei anderen Betriebssystemen auch Software-Interrupts. Sie stellen die grundlegende Schnittstelle zwischen Programmierer und Betriebssystem dar.

- **Bibliotheks-Aufrufe (Sektion 3):** Bibliotheks-Aufrufe sind die Schnittstelle für den normalen High-Level-Programmierer. Es sind Funktionen aus den Programm-Bibliotheken wie zum Beispiel der libc (vgl. C-Programmiersprache).

- **Spezialdateien (Sektion 4):** Die sogenannten Devices werden in */dev* geführt. Es sind die Schnittstellen von real existierenden Geräten (zum Beispiele CD-ROM Laufwerke oder Festplatten) zum Betriebssystem. Dies ist eines der tragenden Elemente von Linux-Systemen. Auf sehr elegante Weise werden die Geräte dabei einfach als normale Dateien angesprochen.

- **Dateiformate und Konventionen (Sektion 5):** In dieser Sektion findet man hauptsächlich Konfigurationsdateien aller Art. Sie wird sicherlich am meisten von Linux-Administratoren verwendet.

- **Spiele (Sektion 6):** In dieser Sektion findet man die Man Pages von Spielen, die von den meisten Linux-Distributionen standardmäßig mitgeliefert werden.

- **Makro-Packages und Konventionen (Sektion 7):** Wenn man als Entwickler kompatibel mit anderen arbeiten will, wird man oft in dieser Gruppe nachsehen, ob bereits ein Konsens auf breiter Basis für ein Problem oder eine Arbeitsweise geschaffen wurde. Dies ist auch die wichtigste Gruppe, wenn man tiefer in die Geheimnisse der Man Pages eindringen will. Denn das Fundament des Man-Formats bildet das Makro-Package „groff“.

- **System Management Kommandos (Sektion 8):** Programme für den Systemverwalter (der sogenannte „root“) , die aber einschneidende Wirkung auf die normalen Benutzer haben. Mehr zu Benutzer und ihren Rechten finden Sie im Kapitel 4 „Nutzerverwaltung“.

- **Kernel Routinen (Sektion 9):** Dies ist keine standardisierte Sektion. Es ist gedacht für Kernel-Entwickler, die neue Routinen testen wollen.

Wenn gleichlautende Einträge in mehreren Sektionen vorkommen, kann man die Suche mit Angabe der Sektion als Parameter einschränken:

`man` *sektion befehl*

Kennt man nicht den genauen Befehl, den man für seine Problemstellung sucht, kann man die Man Pages nicht nur nach einem konkreten Befehl (Eintrag) durchsuchen, sondern auch den Teil „BESCHREIBUNG“ einer Man Page.

`man -k` *begriff*

Es wird dann eine Liste mit Befehlen ausgegeben, die dem Suchbegriff genügen. In Klammern steht bei jedem Befehl die Nummer der Sektion.

3.1.4 X Man Pages

Steht Ihnen eine Desktopumgebung unter Linux zur Verfügung (vgl. X Window System), so können Sie auch über eine grafische Oberfläche (mit dem Programm *xman*) auf die Man Pages zugreifen. Abbildung 3.2 zeigt das Lesen der Man Page zum Befehl `du` mit *xman*.

`xman` lässt sich einfach aus der Kommandozeile starten mit dem Befehl:

```
xman &
```

Durch das „&“ als Parameter wird *xman* als Hintergrundprogramm gestartet und die Kommandozeile steht für weitere Eingaben zur Verfügung. Danach erscheint das kleine Steuerungsfenster (oben rechts in der Abbildung 3.2), über das man mit der Schaltfläche „Help“ eine Hilfetext zur Bedienung von *xman* angezeigt werden kann. Die Schaltfläche „Manual Page“ führt zum größeren Fenster (oben links in der Abbildung), in dem die gewünschte Man Page angezeigt werden kann. *xman* bietet nicht die Möglichkeit den gewünschten Begriff gleich beim Aufruf zu übergeben, wie es bei *man* der Fall ist.

Über den Menüpunkt „Options“ (oben links im „Manual Page“ Fenster) können Sie die Ansicht so ändern, wie sie in der Abbildung zu sehen ist („Show Both Screens“). Der Menüpunkt „Sections“ erlaubt die direkte Auswahl einer Sektion. Im oberen Teilfenster werden dann die entsprechenden Einträge dieser Sektion aufgelistet. Durch einen Maus-Klick auf einen Eintrag wird die dazugehörige Man Page im unteren Teilfenster angezeigt. Das interessanteste Menükommando ist „Search“ unter dem Menüpunkt „Options“. Damit kann rasch eine bestimmte Information gesucht und angezeigt werden.

3.2 Hilfesuche

Am Ende des letzten Abschnitts wurde bereits eine Methode vorgestellt wie man Hilfe zu einer bestimmten Problemstellung findet ohne Kenntnis eines konkreten Befehls (vgl. `man -k`). In diesem

Abschnitt werden noch zwei weitere Befehle vorgestellt, die sehr hilfreich bei der Suche nach Informationen zu einer bestimmten Problemstellung sein können.

3.2.1 Die Befehle whatis und apropos

whatis ist besonders gut geeignet für die Suche nach Informationen, falls man schon einen Begriff zur Problemstellung weiß, jedoch nicht sicher ist, ob der Befehl, etc. der richtige Schritt in Richtung Problemlösung ist:

`whatis` *begriff*

Es wird eine kurze Beschreibung (meist einzeilig) des angegebenen Begriffes ausgegeben. *whatis* greift auf den NAME-Abschnitt innerhalb der Man Pages zurück und durchsucht diesen nach dem gewünschten Begriff. Folglich ist der *whatis*-Befehl auf die Man Pages als Grundlage angewiesen. Der Nachteil von *whatis* ist, dass es nur exakte Übereinstimmungen mit dem Suchbegriff zurückgibt.

Der Befehl *apropos* ermöglicht eine breitere Suche nach Informationen. Er gibt auch kurze Beschreibungen bei Teilübereinstimmungen zurück, birgt jedoch in sich den Nachteil bei relativ allgemeingültigen Begriffen (zum Beispiel „conf“), dass eine Vielzahl von Kurzbeschreibungen zurückgegeben werden. Die Grundlage für die Suche sind ebenfalls die Man Pages:

`apropos` *thema*

3.2.2 Der info Befehl

info ist ein wesentlich leistungsstärkeres Programm als *man*. Es eignet sich besonders zur Darstellung umfangreicherer Hilfetexte, weil es mit Querverweisen über mehrere Dateien zurecht kommt. Dementsprechend findet man auch eine ausführlichere Dokumentation mit dem *info*-Befehl als die, wie sie in den Man Pages vorliegt. Die info-Dokumentationen liegen dementsprechend in eigenen, von den Man Pages unabhängigen Format vor (info-Format), welches von der Free Software Foundation (FSF) bereitgestellt wird. Es ist das bevorzugte Dokumentationsformat für alle GNU-Werkzeuge (zum Beispiel die Befehle *find* und *grep*). Abbildung 3.3 zeigt die info-Dokumentation zum Befehl *du*.

Der *info*-Befehl wird wie folgt aufgerufen:

`info` *thema*

Die Dokumentation wird in der Vollansicht innerhalb der Kommandozeile angezeigt. Mit der Leer-

taste und der „Backspace“ Taste kann man innerhalb der info-Dokumentation nach unten beziehungsweise nach oben scrollen. Mit der „Tabulator“ Taste und der „Return“ Taste kann man den nächsten Querverweis (erkenntlich durch ein vorangestelltes „*“ Zeichen) selektieren und diesen dann zu einem anderen info-Dokument verfolgen.

3.3 HOWTOs

In beiden vorangegangenen Abschnitten wurde die Suche nach Informationen beziehungsweise Hilfetexten zu einer einfachen Problemstellung beschrieben (zum Beispiel „Suchen einer Datei“ - `man find`). Wie findet man aber Hilfe zu komplexeren Problemstellungen wie zum Beispiel die Installation und Konfiguration ein HTTP Servers unter Linux (vgl. Apache HTTP Server Project).

Für diesen Zweck stehen sogenannte HOWTO-Texte zur Verfügung. Dabei handelt es sich um systematisch aufgebaute Grundlagentexte (meist im HTML-Format) zu bestimmten Themen. Zu nahezu jeder Problemstellung, vor die ein Benutzer oder Administrator eines Linux-Systems früher oder später gestellt wird, gibt es einen HOWTO-Text:

- Installation von spezieller Hardware (SCSI-Festplatten, CD-ROM-Laufwerken, Netzwerkkarten, Tastaturen, Mäusen, etc.)
- Installation spezieller Software (Server-Dienste: HTTP Server, Application Server, Datenbank Server, etc.)
- Kombination und Integration verschiedener Software
- u.v.m.

Insgesamt beträgt der Umfang der HOWTO-Texte ein Vielfaches dieses Kurses. Die HOWTO-Dateien befinden sich zumeist in */usr/doc/howto* oder in */usr/share/doc/howto*. Die aktuellen Versionen finden Sie im Internet auf der Seite des Linux Documentation Project (LDP). Innerhalb des LDP sind aus manchen HOWTO-Texten, die ursprünglich nur in elektronischer Form verfügbar waren, ganze Bücher entstanden, die Sie in einer Buchhandlung erwerben können. Um das Beispiel am Anfang dieses Abschnittes aufzugreifen, finden Sie relativ schnell auf der Seite des LDP eine sehr ausführliche Dokumentation zur Installation und Konfiguration des Apache HTTP Servers (`http://www.tldp.org/HOWTO/Apache-Overview-HOWTO.html`).

```
DU(1)                      User Commands                      DU(1)



NAME
       du - estimate file space usage

SYNOPSIS
       du [OPTION]... [FILE]...

DESCRIPTION
       Summarize  disk usage of each FILE, recursively for direc-
       tories.

       Mandatory arguments to  long  options  are  mandatory  for
       short options too.

       -a, --all
              write counts for all files, not just directories

       --apparent-size
              print  apparent  sizes,  rather  than  disk  usage;
              although the apparent size is usually  smaller,  it
              may  be  larger  due  to holes in (`sparse') files,
              internal fragmentation, indirect  blocks,  and  the
              like

       -B, --block-size=SIZE use SIZE-byte blocks

       -b, --bytes
              equivalent to `--apparent-size --block-size=1'

       -c, --total
              produce a grand total

       -D, --dereference-args
              dereference FILEs that are symbolic links

       -h, --human-readable
              print sizes in human readable format (e.g., 1K 234M
              2G)

 Manual page du(1) line 1/119 32%
```

Abbildung 3.1: Man Page zum Befehl du

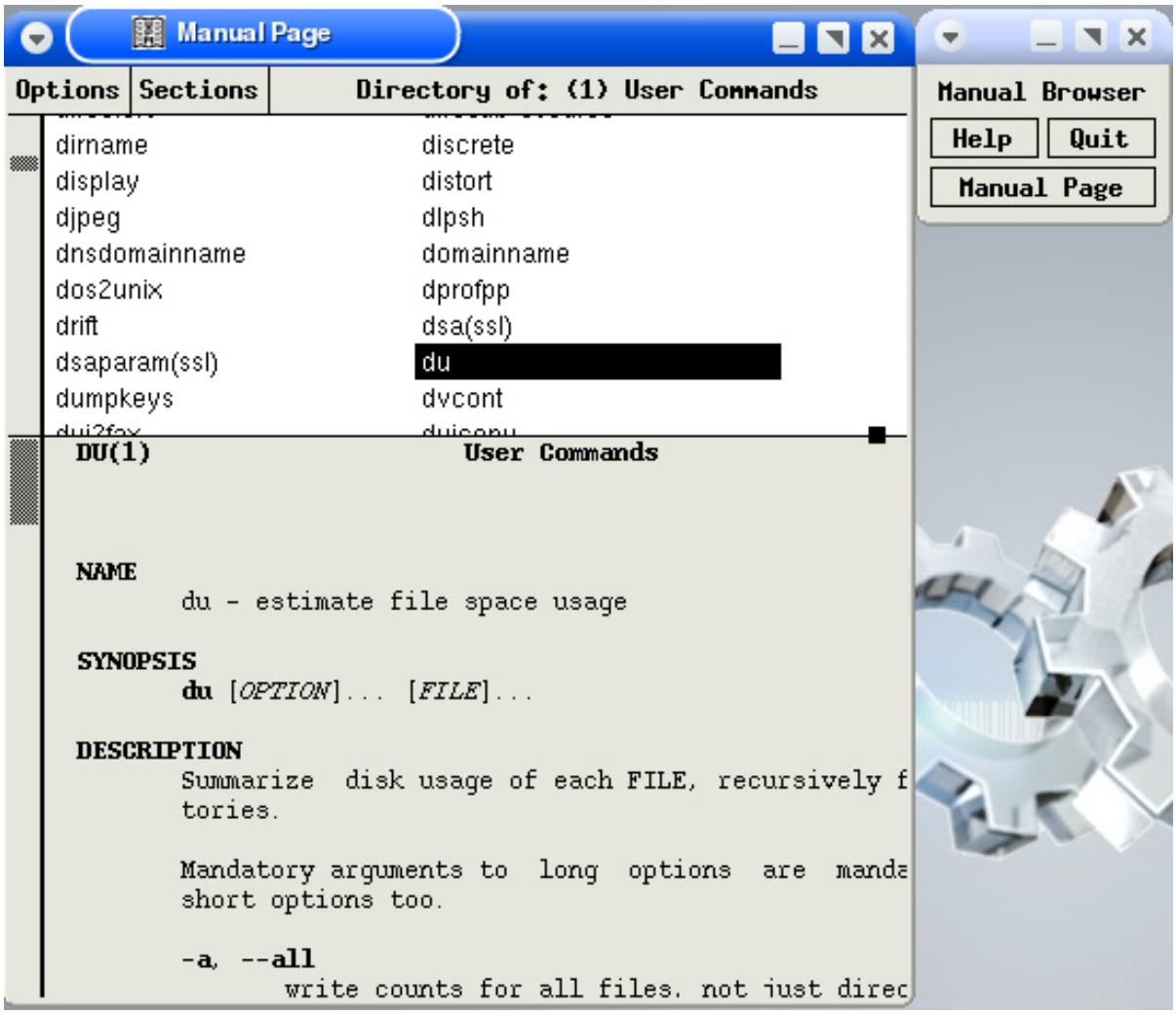

Abbildung 3.2: Man Page zum Befehl du unter xman

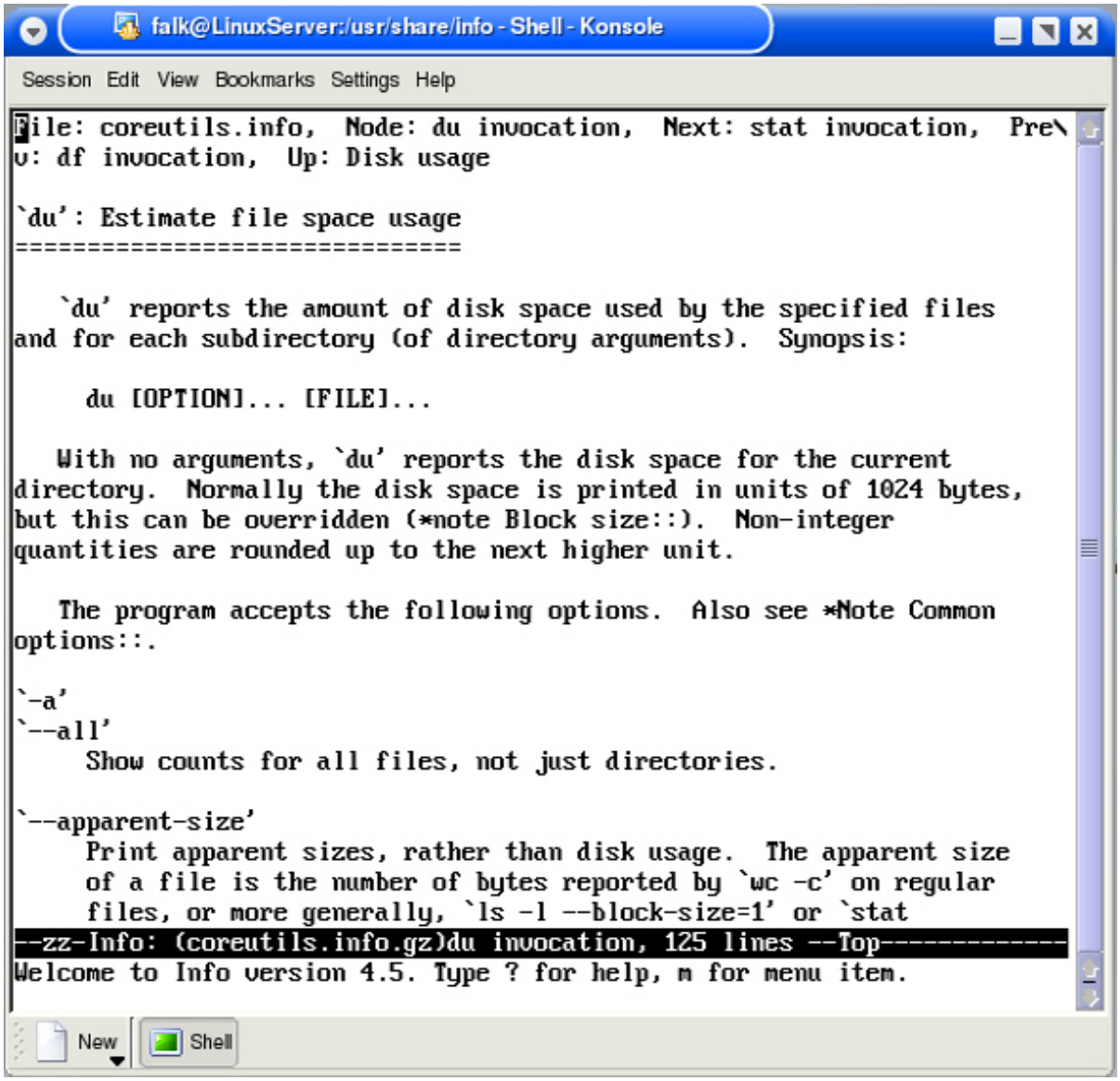
falk@LinuxServer:/usr/share/info - Shell - Konsole
Session Edit View Bookmarks Settings Help
File: coreutils.info, Node: du invocation, Next: stat invocation, Prev: df invocation, Up: Disk usage
`du': Estimate file space usage
`du' reports the amount of disk space used by the specified files and for each subdirectory (of directory arguments). Synopsis:
du [OPTION]... [FILE]...
With no arguments, `du' reports the disk space for the current directory. Normally the disk space is printed in units of 1024 bytes, but this can be overridden (*note Block size::). Non-integer quantities are rounded up to the next higher unit.
The program accepts the following options. Also see *Note Common options::.
`-a'
`--all'
Show counts for all files, not just directories.
`--apparent-size'
Print apparent sizes, rather than disk usage. The apparent size of a file is the number of bytes reported by `wc -c' on regular files, or more generally, `ls -l --block-size=1' or `stat
--zz-Info: (coreutils.info.gz)du invocation, 125 lines --Top----------
Welcome to Info version 4.5. Type ? for help, m for menu item.
New
Shell

Abbildung 3.3: info Dokumentation zum Befehl du

4 Nutzerverwaltung

Dieses Kapitel erklärt Ihnen die Grundlagen der Nutzerverwaltung. Neben den Aspekten eines Multi-User Systems werden die unterschiedlichen Arten von Benutzern unter Linux erläutert. Außerdem erfahren Sie, wie Sie Benutzer und Benutzergruppen anlegen und verwalten können.

Die folgenden Punkte sind Inhalt dieses Kapitels.

- 4.1 Einleitung / Multi User System
- 4.2 Benutzer
- 4.3 Benutzergruppen
- 4.4 Benutzerklassen / Berechtigungsklassen
- 4.5 Benutzerdateien

4.1 Multi User System

Linux ist ein Multi-User System. Dies bedeuted, dass an einem System mehrere Benutzer gleichzeitig arbeiten können. Dies mag auf den ersten Blick etwas seltsam erscheinen da ja in der Regel an einem Rechner Peripheriegeräte wie Tastatur, Maus und Monitor nur einmal vorhanden sind. Allerdings können sich beliebig viele Nutzer z.B. über ein Netzwerk an einem Linux System anmelden und auf diesem arbeiten. Zudem werden auf einem Linux-System eine Vielzahl von Prozessen unter einer anderen Benutzerkennung gestartet als der, unter welcher man sich selbst angemeldet hat (siehe Abbildung 4.1).

Ein Multi-User System setzt natürlich ein geeignetes Berechtigungssystem voraus um zum Einen das System selbst vor den Handlungen seiner Nutzer, zum Anderen die Nutzer vor Handlungen anderer Nutzer zu schützen.

Auf einem Linuxsystem können Benutzer und Benutzergruppen angelegt und verwaltet werden. Ein Benutzer kann hierbei Mitglied einer oder mehrer Benutzergruppen sein. Rechte können sowohl Benutzergruppen als auch einzelnen Benutzern zugewiesen werden.

```
alex@work:~> ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    620   256 ?        S    Jun12   0:06 init [3]
root         2  0.0  0.0      0     0 ?        SW   Jun12   0:00 [keventd]
root         3  0.0  0.0      0     0 ?        SW   Jun12   0:00 [kapmd]
root         4  0.0  0.0      0     0 ?        SWN  Jun12   0:00 [ksoftirqd_CPU0]
root         5  0.0  0.0      0     0 ?        SW   Jun12   0:00 [kswapd]
root         6  0.0  0.0      0     0 ?        SW   Jun12   0:00 [bdflush]
root         7  0.0  0.0      0     0 ?        SW   Jun12   0:00 [kupdated]
root         8  0.0  0.0      0     0 ?        SW   Jun12   0:00 [kinoded]
root         9  0.0  0.0      0     0 ?        SW   Jun12   0:00 [mdrecoveryd]
root        13  0.0  0.0      0     0 ?        SW   Jun12   0:00 [kjournald]
root        58  0.0  0.0      0     0 ?        SW<  Jun12   0:00 [lvm-mpd]
root        99  0.0  0.0      0     0 ?        SW   Jun12   0:00 [kjournald]
root       100  0.0  0.0      0     0 ?        SW   Jun12   0:00 [kjournald]
root       389  0.0  1.3   7252  4168 ?        S    Jun12   0:00 /usr/local/apache
nobody     473  0.0  1.3   7252  4180 ?        S    Jun12   0:00 /usr/local/apache
nobody     474  0.0  1.3   7252  4180 ?        S    Jun12   0:00 /usr/local/apache
nobody     475  0.0  1.3   7252  4180 ?        S    Jun12   0:00 /usr/local/apache
nobody     476  0.0  1.3   7252  4180 ?        S    Jun12   0:00 /usr/local/apache
nobody     477  0.0  1.3   7252  4180 ?        S    Jun12   0:00 /usr/local/apache
root       506  0.0  0.0      0     0 ?        SW   Jun12   0:00 [eth0]
root       623  0.0  0.1   1560   624 ?        S    Jun12   0:00 /sbin/syslogd
root       626  0.0  0.4   2360  1408 ?        S    Jun12   0:00 /sbin/klogd -c 1
postfix  13660  0.0  0.4   4204  1300 ?        S    02:55   0:00 pickup -l -t fifo
root     24336  0.3  0.7   5420  2320 ?        S    04:31   0:00 sshd: root@pts/2
root     24337  0.0  0.7   5420  2320 ?        S    04:31   0:00 sshd: root@pts/2
root     24339  0.0  0.5   2836  1644 pts/2    S    04:31   0:00 -bash
alex     24374  0.0  0.2   2668   716 pts/0    R    04:33   0:00 ps aux
alex@work:~>
```

Abbildung 4.1: Beispiel mehrere Prozesse mit unterschiedlichen Nutzerkennungen. (Kommando: ps aux)

4.2 Benutzer

4.2.1 Eigenschaften eines Benutzers

Für den Nutzer eines Linux-Systems besteht ein Benutzer auf den ersten Blick lediglich aus seinem Benutzername und Passwort. Das ist für die tägliche Arbeit auch ausreichend. Das System jedoch verwaltet noch eine Reihe weiterer Informationen.

Linux verwendet intern eine **Benutzer-ID** zur Verwaltung eines Benutzers. Die eigene Benutzer-ID (*UID*) kann einfach mit dem Befehl

```
id -u
```

abgefragt werden. Als Ausgabe erscheint die vom System an den Benutzer vergebene UID. Linux verwendet intern immer in erster Linie die UID um Resourcen zu verwalten die einem Benutzer zugeordnet sind. Zusätzlich zur UID ist jedem Benutzer noch eine **Gruppen-ID**(*GID*) zugewiesen.

Diese kann durch den Befehl

```
id -g
```

abgerufen werden.

Neben der Benutzer-ID und der Gruppen-ID ist jedem Benutzer ein **Heimatverzeichnis** zugeordnet. Alle Unterverzeichnisse und Dateien in diesem Heimatverzeichnis gehören diesem Benutzer. Hier können Dateien abgelegt werden welche meist ausschließlich von diesem Benutzer genutzt werden. Außerdem werden hier Dateien gespeichert die beispielsweise spezifische Einstellungen für Applikationen beinhalten.

Jedem Benutzer kann zudem noch eine Arbeitsumgebung zugeordnet werden, welche beim Anmeldevorgang geladen wird (Login Programm). Dies kann eine Shell oder auch ein Display-Manager sein.

4.2.2 Benutzertypen

Auf einem Linux-System gibt es drei verschiedene Arten von Benutzertypen: **root** (Systemverwalter, Superuser), **Systembenutzer** und **Standardbenutzer**.

root

Der Benutzer *root* wird auch als **Superuser** bezeichnet. Dem Benutzer *root* ist stets die UID 0 zugeordnet. Er verfügt über alle Rechte im System, kann somit auch sämtliche Änderungen an der Konfiguration vornehmen. Es gibt Prozesse (Programme) in einem Linux-System die stets mit der Kennung von *root* ausgeführt werden. Ein Beispiel hierfür ist der Prozess *init*. Der Prozess *init* ist der erste Prozess der stets nach dem Booten eines Linux-System als erster ausgeführt wird und die Kontrolle übernimmt. *init* initialisiert alle weiteren Prozesse. Um dies zu gewährleisten benötigt es weitreichende Rechte und wird deshalb mit der Kennung des Benutzers *root* ausgeführt. Weiteres zum init-Prozess finden Sie in Kapitel 6.6 „`init.d`“.

Die Tatsache das *root* über alle Rechte verfügt hat nicht nur Vorteile. Man kann eine Installation nicht nur administrieren, sondern auch zerstören wenn man als *root* arbeitet. Aus diesem Grund empfiehlt es sich stets die Alltagsarbeit als *Standarduser* zu erledigen, und nur dann wenn es die Aufgabe erfordert einen Benutzerwechsel zu *root* durchführen mittels des Befehls `su -`. Benutzt man einen Display-Manager so kann man beispielsweise einfach eine neue Konsole öffnen und sich in dieser als *root* anmelden.

Systembenutzer

Nicht jeder Prozess der beispielsweise beim Starten eines Linux-Systems geladen werden soll muss unter der Kennung von *root* ausgeführt werden. Dies erhöht auch die Sicherheit des Systems, da dadurch die Möglichkeiten für Missbrauch eingeschränkt werden.

Derartige Prozesse werden mit der Kennung eines *Systembenutzer* gestartet. Ein Beispiel hierfür ist der Benutzer *nobody*. *nobody* wird stets von Prozessen als Kennung verwand wenn nur minimale Rechte notwendig sind (z.B. Apache). Ein anderes Beispiel ist der Benutzer *mysql*, welcher von einer MySql Datenbank verwendet wird.

Allerdings bleibt festzuhalten das der Unterschied zwischen *Systembenutzern* und *Standardbenutzern* nicht durch das Linux-Rechtesystem an sich vorgegeben ist. Es handelt sich hier eher um ein Konzept wie mit den Möglichkeiten der Rechtevergabe unter Linux gearbeitet wird. So wird Systembenutzern meist keine Shell zugeordnet so das ein normaler Login erst gar nicht möglich ist.

Standardbenutzer

Normale Benutzerkonten mit denen normale Benutzer arbeiten.

4.2.3 Benutzer verwalten

Um die Benutzer auf einem Linux-System verwalten zu können müssen Sie über die entsprechenden Rechte im System verfügen. Auf Standardsystemen benötigen Sie ein Login als Benutzer *root*. Sie können zur Verwaltung der Benutzer diverse Tools Ihrer Linux Distribution nutzen, aber natürlich gibt es auch hierfür Befehle die in der Shell ausgeführt werden können. Einige grundlegende hiervon sollen hier kurz erläutert werden.

Benutzer anlegen

Zum anlegen eines neuen Benutzers steht Ihnen das Kommando *useradd* zur Verfügung.

```
useradd alexander -p test -d /home/alexander -m -g users
```

Dieser Befehl legt einen neuen Benutzeraccount für den Benutzer *Alexander* an. Hier eine kurze Erklärung der verwendeten Parameter:

- **-p** gibt das Passwort an, in diesem Fall test

- **-d** gibt das Heimatverzeichnis an, in diesem Fall */home/alexander*
- **-m** gibt an, dass das Heimatverzeichnis erstellt werden soll
- **-g** gibt die Benutzergruppe an in welcher der neue Benutzer Mitglied ist, in diesem Fall *users*

Mehr Parameterinformationen mit: `man useradd`

Benutzer modifizieren

Um Benutzerinformationen nachträglich ändern zu können gibt es das Kommando *usermod*.

```
usermod alexander -d /home/alexanderNeu -m -g root
```

Der Benutzer *alexander* hat nun als neues Heimatverzeichnis das Verzeichnis */home/alexanderNeu* und ist Mitglied in der Gruppe *root*.

Mehr Parameterinformationen über: `man usermod`

Benutzer löschen

Um einen Benutzer zu löschen verwenden Sie das Kommando *userdel*.

```
userdel alexander
```

Der Benutzer *alexander* ist nun gelöscht.

Mehr Parameterinformationen über: `man userdel`

Passwort ändern

Natürlich können Sie das Passwort auch mit *usermod* ändern. Allerdings benötigen Sie hier wie bereits erwähnt die nötigen Berechtigungen. Sind Sie als normaler Benutzer angemeldet so können Sie Ihr Passwort mit Hilfe von *passwd* ändern.

```
passwd
```

Sie werden zur Eingabe des neuen Passwortes aufgefordert.

4.3 Benutzergruppen

Wie bereits erwähnt können auf einem Linux-System beliebig viele Benutzergruppen angelegt und verwaltet werden. Die meisten Linux-Distributionen legen bereits bei der Installation eine Reihe von Benutzergruppen an. So finden sich in der Regel immer die Benutzergruppen *root* und *users*. Mitglied von *root* sind Benutzer die über root Berechtigungen verfügen. In der Gruppe *users* sind alle normalen Benutzer beherbergt.S

Benutzergruppen erleichtern die Administration von Benutzern ungemein, da mit Ihrer Hilfe die Zugriffsrechte einer ganzen Anzahl von Benutzern gezielt verändert werden kann. So können zum Beispiel Benutzer die alle Zugriff auf eine Anzahl von Dateien für die Arbeit an einem gemeinsamen Projekt benötigen Mitglied einer Benutzergruppe sein über welche die Zugriffsrechte gesteuert werden.

Mit dem Kommando

```
groups
```

können Sie sich die Benutzergruppen anzeigen lassen in denen der Benutzer, mit dem Sie angemeldet sind, Mitglied ist.

4.3.1 Benutzergruppen verwalten

Wie bei der Verwaltung von Benutzern stellen auch bei der Verwaltung von Benutzergruppen die meisten Distributionen eigene Tools zur Verfügung. An dieser Stelle wird wieder auf die Kommandos eingegangen die über die Shell ausgeführt werden können. Auch zur Verwaltung der Benutzergruppen muss man natürlich über die entsprechenden Rechte verfügen.

Benutzergruppe anlegen

Mit dem Kommando *groupadd* können neue Benutzergruppen angelegt werden. Der Befehl

```
groupadd ba
```

Fügt dem System die Benutzergruppe *ba* hinzu.

Benutzergruppe modifizieren

Mit dem Befehl

```
groupmod ba -n ba_stuttgart
```

ändert den Namen der Benutzergruppe *ba* in *ba_stuttgart*.

Benutzergruppe löschen

Mit dem Befehl

```
groupdel ba_stuttgart
```

wird die Benutzergruppe *ba_stuttgart* wieder gelöscht.

Bitte beachten Sie das auf keinen Dateien bzw. Verzeichnisesn mehr die Gruppen-ID der zu löschenden Benutzergruppe verknüpft sein darf. Zudem darf kein Benutzer im System die zu löschende Benutzergruppe als Primärgruppe verwenden. In beiden Fällen würde der Löschvorgang scheitern.

4.4 Benutzerklassen / Berechtigungsklassen

Um den Zugriff auf Dateien und Ressourcen zu regeln gibt es unter Linux sogenannte Benutzer- und Berechtigungsklassen. Es werden die drei Benutzerklassen **user**, **group** und **others** unterschieden. Diese Benutzerklassen teilen sich jeweils wieder in Berechtigungsklassen für Lese-, Schreib- und Ausführrechte auf.

- **user-read** (Leserecht für Benutzer)
- **user-write** (Schreibrecht für Benutzer)
- **user-execute** (Ausführrecht für Benutzer)
- **group-read** (Leserecht für Benutzergruppe)
- **group-write** (Schreibrecht für Benutzergruppe)
- **group-execute** (Ausführrecht für Benutzergruppe)
- **other-read** (Leserecht für alle Benutzer)
- **other-write** (Schreibrecht für alle Benutzer)
- **other-execute** (Ausführrecht für alle Benutzer)

Dies bedeuted das für eine Datei jeweils Lese-, Schreib- und Ausführrechte für einen Benutzer (user), eine Benutzergruppe (group) und im Bedarfsfall für alle Benutzer (others) gesetzt werden können.

Weiter Informationen wie Sie die hier beschriebenen Rechte im konkreten Fall setzen können finden sie im Kapitel 5.5 Zugriffsrechte.

4.4.1 Berechtigungsklassen

Im folgenden soll kurz auf die Bedeutung der einzelnen Berechtigungsklassen eingegangen werden.

Lesen

Ist das Leserecht für eine *Datei* gesetzt so kann diese durch ein Programm geöffnet und interpretiert (gelesen) werden. Zum Ausführen von Programmen welche ja letztendlich auch in einer *Datei* gespeichert sind muss das Leserecht nicht gesetzt sein. Hierzu reicht das Recht zum Ausführen.

Verzeichnisse nehmen hier eine gewisse Sonderrolle ein. Da sie lediglich die Informationen über die in ihnen enthaltenen Dateien und Unterverzeichnisse enthalten, regelt hier das Leserecht ob diese Informationen z.B. durch den Befehl

```
ls
```

ausgelesen werden dürfen oder nicht. Allerdings können die Informationen einer Datei, welche sich in einem Lesegeschützten Verzeichnis befindet bei bekannten Dateinamen immer noch abgerufen werden. Das Leserecht für Verzeichnisse ist also unabhängig von den Rechten der sich darin befindenten Dateien.

Schreiben

Das Schreibrecht regelt wie aus dem Name schon zu erkennen ist, ob Dateien/ Resourcen modifiziert oder gelöscht werden dürfen.

Ausführen

Das Ausführen von Programmen kann mit diesem Recht kontrolliert werden. Je nachdem ob der Benutzer das Recht zum Ausführen besitzt kann das Programm ausgeführt werden.

Auch hier spielen *Verzeichnisse* eine gewisse Sonderrolle. Ist das Ausführrecht für ein Verzeichnis nicht gesetzt so können keine Informationen über Dateien ausgegeben werden die sich in dem Verzeichnis befinden.

4.5 Benutzerdateien

In diesem Abschnitt soll kurz darauf eingegangen werden, wo die in diesem Kapitel beschriebenen Informationen innerhalb von Linux abgelegt werden.

Linux legt alle Benutzer- und Benutzergruppen Informationen in Dateien ab. Im Einzelnen sind dies die Dateien *passwd*, *shadow* und *groups*. Sie finden die Dateien im Verzeichnis */etc*.

4.5.1 Die Datei passwd

In der Datei */etc/passwd* sind alle Benutzerdaten gepeichert. Die einzelnen Benutzer werden in folgenden Format abgespeichert:

```
Benutzername : Passwort : UID : GID : Info : Heimatverzeichnis : Shell
```

- **Benutzername:** Benutzername (Login)
- **Passwort:** bei älteren System das Passwort verschlüsselt, bei neuen System ein *x* welches darauf hinweist das das Passwort in der Datei */etc/shadow* gespeichert ist
- **UID:** User-ID des Benutzers (muss größer als 100 und kleiner als 64000 sein)
- **GID:** Gruppen-ID
- **Info:** Platz für zusätzliche Informationen zum Benutzer
- **Heimatverzeichnis:** Heimatverzeichnis des Benutzers
- **Shell:** Shell die nach Anmeldung gestartet werden soll

4.5.2 Die Datei shadow

Wie bereits erwähnt wurde, wurden bei älteren Linux-Systemen die Passwörter direkt in der Datei */etc/passwd* abgelegt. So verwaltete Passwörter sind jedoch leicht mit einer Wörterbuchatacke zu überwinden. Aus diesem Grund werden Passwörter in aktuellen Linux Distributionen in der Datei */etc/shadow* abgelegt und durch ein spezielles System besser geschützt.

Die Passwörter werden im folgenden Format abgelegt.

```
Benutzername : Passwort : DOC : MinD : MaxD : Warn : Exp : Dis : Res
```

- **Benutzername:** Benutzername (Login)
- **Passwort:** verschlüsseltes Passwort, * oder ! bedeuten kein Passwort
- **DOC:** Tag der letzten Passwortänderungen (Format: Tage seit 1.1.1970)
- **MinD:** Tage die das Passwort minimal gültig ist
- **MaxD:** Tage die das Passwort maximal gültig ist
- **Warn:** Anzahl der Tage die vor Ablauf des Passwortes ab der gewarnt werden soll
- **Exp:** Anzahl Tage die das Passwort trotz Verfall noch gültig ist
- **Dis:** Bis zu diesem Tag ist das Benutzerkonto gesperrt (Format: Tage seit 1.1.1970)
- **Res:** Im Augenblick unbenutzt

4.5.3 Die Datei groups

Zusätzlich zu der in der Datei */etc/passwd* für einen Benutzer definierten Benutzergruppe kann der Benutzer in der Datei */etc/groups* weiteren Benutzergruppen zugeordnet werden.

Die Einträge erfolgen in Format:

```
Gruppenname : Passwort : GID : Benutzernamen
```

- **Gruppenname:** Name der Benutzergruppe
- **Passwort:** wird ein Passwort angegeben so können auch Benutzer die nicht ausdrücklich Mitglied der Gruppe sind durch Passworteingabe Zugang bekommen
- **GID:** Gruppen-ID
- **Benutzername:** Mitglieder der Benutzergruppe (durch Komma getrennt)

5 Dateiverwaltung

Linux basiert auf einem hierarchischen, aber auch sehr flexibel handhabbaren Dateisystem. Im nun folgenden Kapitel wird die Dateisystemhierarchie von Linux auf Basis des Filesystem Hierarchy Standard (FHS) vorgestellt, insbesondere auch der Umgang mit Dateien, Links, Zugriffsrechten, etc. Für vertiefende Informationen über den FHS empfiehlt sich `http://www.pathname.com/fhs/`.

Die folgenden Punkte sind Inhalt dieses Kapitels; die hier genannten Befehle stellen nur eine Übersicht über die wichtigsten Befehle der nachfolgenden Kapitel dar. In den jeweiligen Kapiteln werden auch weitere wichtige Befehle aufgezeigt.

- 5.1 Dateisystemhierarchie (FHS)
- 5.2 Dateisysteme (ext2, ext3, ReiserFS, AFS,...)
- 5.3 Befehle im Zusammenhang mit Dateien und Verzeichnissen
- 5.4 Verweise: Symbolische- und Hard-Links (Befehl: *ln*)
- 5.5 Zugriffsrechte (Befehle: *chmod, chgrp, chown*)
- 5.6 Einbinden von Dateisystemen (Befehl: *mount, umount*)
- 5.7 Partitionierung (Befehl: *fdisk, mkfs*)
- 5.8 Dateisystem-Integrität (Befehl: *df, du, fsck*)

5.1 Dateisystemhierarchie

5.1.1 Hierarchischer Aufbau

Der Aufbau der Linux-Dateisystemhierarchie ist eine Baumstruktur mit definierter hierarchischer Gliederung. Aufgrund dieser Baumstruktur wird diese Hierarchie auch kurz *Dateibaum* genannt. Dies ist auch einer der größten Unterschiede zum Windows-Dateisystem: es gibt keine Laufwerksbuchstaben bei Linux. Unterschieden werden

- **(Reguläre) Dateien** sind „normale“ Dateien im Text- oder Binärformat welche mit Standard-Befehlen les-, veränder- und schreibbar sind.
- **Verzeichnisse** können Dateien oder Gerätedateien beinhalten.
- **Gerätedateien** dagegen sind Dateien mit welchen im Linux-System installierte Geräte (z.B. Festplatten, Schnittstellen, ...) angesprochen werden können. So ist z.B. */dev/hda* die Gerätedatei für die erste am primären IDE-Bus installierte Festplatte. Über diese Gerätedatei kann die Festplatte als *Blockdevice* (also als Blockorientiertes Gerät) angesprochen werden.

Ein großer Unterschied zu Windows-Dateisystemen ist die Tatsache, dass Datei- und Verzeichnisnamen Case-Sensitive gehandelt werden. Befinden sich beispielsweise die drei Dateien *tvprogramm.txt*, *TVprogramm.txt* und *tvProgramm.txt* in ein und demselben Verzeichnis, so handelt es sich um drei unterschiedliche Dateien. Angesprochen werden all diese Dateien und Verzeichnisse über *absolute* oder *relative* Datei- bzw. Pfadnamen.

Pfade

- **absolute Pfade:** ausgehend von der Wurzel des Dateibaums (auch „root“ Verzeichnis genannt, Abbildung 5.1 links oben „/“) geben diese Pfade einen vollständigen Weg an, welche Datei oder Verzeichnis gemeint ist. Jedes Objekt im Dateibaum kann so eindeutig angesprochen werden. Beispiel für einen absoluten Pfad welcher auf die Datei *Kap1.tex* von Benutzer *anja* zeigt ist */home/anja/Texte/Kap1.tex*.
- **relative Pfade:** ausgehend vom aktuellen Verzeichnis in welchem man sich befindet wird ein Pfad angegeben. Befinden wir uns z.B. im Moment im Unterverzeichnis *Texte* des Benutzers *anja* und wollen auf die Datei *a.out* Verweisen, so ist der Pfad *../Prog/a.out*.

Es gibt Abkürzungen für bestimmte Verzeichnisse:

- $\sim$ steht für das Home-Verzeichnis
- **.** bezeichnet das aktuelle Verzeichnis
- **..** übergeordnetes Verzeichnis

Man kann also ohne sein Verzeichnis zu verlassen Dateien im gesamten Dateibaum ansprechen. Die einzelnen Elemente dieser Pfadangaben werden jeweils durch einen „/“ („slash“) abgetrennt. Um sich Tipparbeit auf der Kommandozeile zu ersparen, kann man mit der **Tabulator-Taste** jederzeit automatisiert Datei- oder Verzeichnisnamen ergänzen.

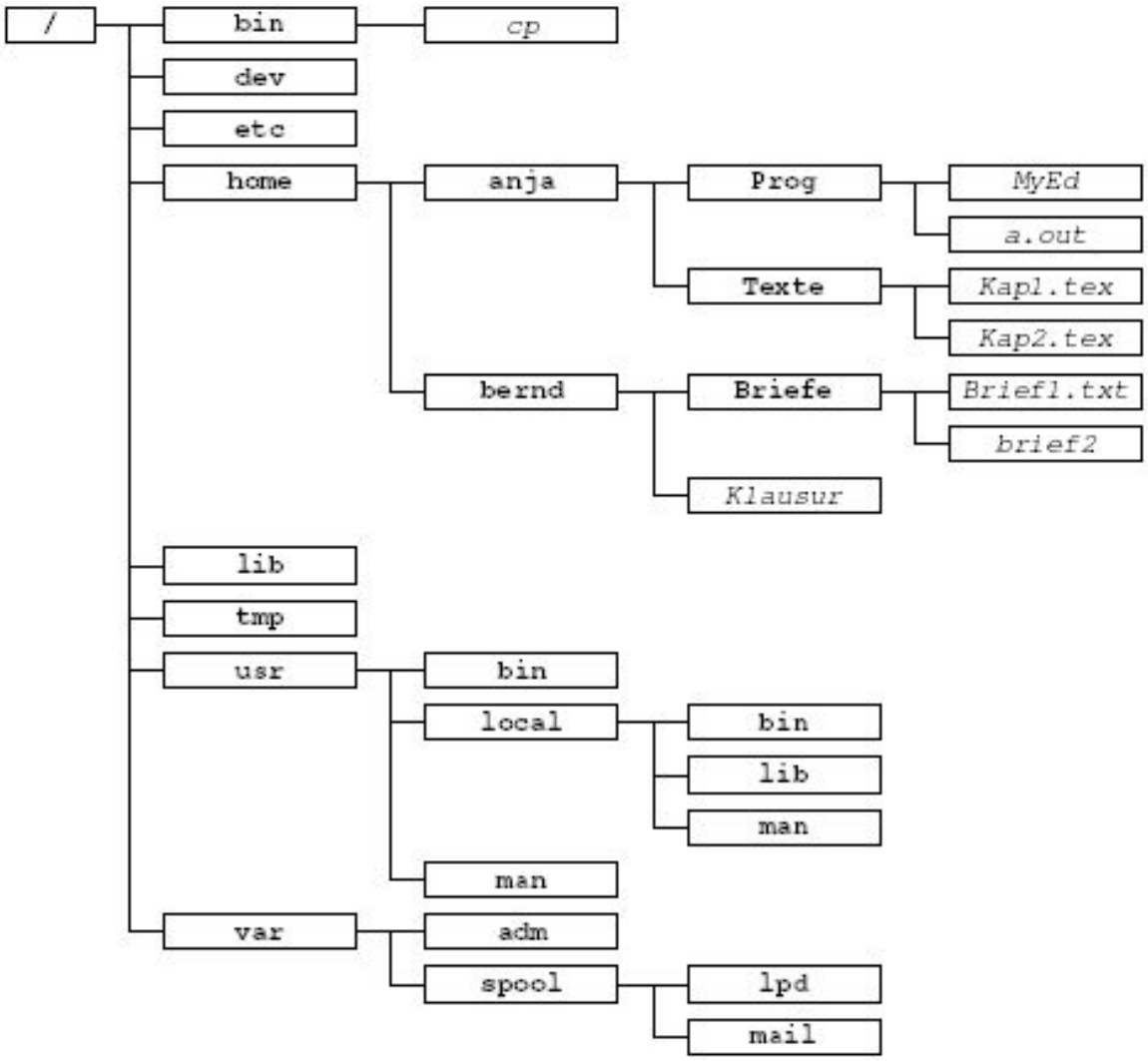

Abbildung 5.1: Verzeichnisbaum eines Linuxsystems [20]

Ein Beispiel hierzu: befindet man sich im Verzeichnis *anja* und will in das Unterverzeichnis *Texte* wechseln, so genügt es den Befehl `cd T` einzugeben und danach die Tabulator-Taste zu drücken. Automatisch wird erkannt, dass es sich hier nur um das Unterverzeichnis *Texte* handeln kann. Der Befehl auf der Kommandozeile wird also ergänzt und es steht somit `cd Texte` auf dem Bildschirm. Ein anschließendes Bestätigen mit der Enter-Taste wechselt in das Unterverzeichnis *Texte*.

Datei- und Verzeichnisnamen

Datei- und Verzeichnisnamen können sehr lang werden und dürfen fast alle Zeichen außer „/“ enthalten. Jedoch gibt es einige Sonderzeichen (Abbildung 5.2, welche in der Shell eine besondere Bedeutung haben. Man sollte vermeiden diese zu benutzen. Falls ein Datei- oder Verzeichnis-

name eines der folgenden Zeichen enthält, kann es schwierig werden Befehle mit diesen Namen auszuführen (z.B. löschen der Datei).

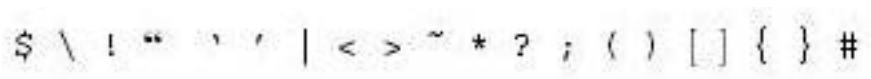

Abbildung 5.2: Sonderzeichen [20]

5.1.2 File Hierarchie Standard (FHS)

Der FHS wird von allen erhältlichen Linux-Distributionen umgesetzt und beschreibt, an welcher Stelle sich welche Dateien und Verzeichnisse im Dateibaum befinden, und wie diese benannt sind.

Der Standard wurde 1993 von der Linux-Community definiert, um proprietären Entwicklungen das Dateisystem und den Baum betreffend zu vermeiden. Eine Organisation, welche den Standard in irgendeiner Form betreut, gibt es nicht. Bei diesem Standard werden alle Dateien in verschiedene Klassen eingeteilt:

1. Data Sharing (Teilen von Daten):
   - **Shareable:** Diese Dateien können von verschiedenen Rechnern im Netzwerk gleichzeitig benutzt werden. Lokal auf einem Rechner handelt es sich um Dateien wie Benutzerdaten, viele ausführbare Programme und gemeinsame Konfigurationsdateien (z.B. */etc/hosts*).
   - **Non-Shareable:** Dateien können nicht von anderen Rechnern benutzt werden, so z.B. Passwort-Dateien, Netzwerkkonfigurationsdateien und System-Logs.
2. Data Modification (Ändern von Daten):
   - **Variable:** Variable Daten können jederzeit geändert werden, so z.B. Logfiles wie */var/log/messages*
   - **Static:** Statische Daten sind Daten, die über einen längeren Zeitraum konsistent bleiben, so z.B. Systembefehle wie ls oder die bash. Diese Dateien ändern sich typischerweise nur wenn z.B. ein Update des Linuxsystems durchgeführt wird.

Systemverzeichnisse

Die meisten Systemverzeichnisse sind definiert und haben entweder eine besondere Bedeutung, oder beinhalten spezifische Daten. Die hier aufgeführte Liste gibt auszugsweise einen Überblick über die wichtigsten Verzeichnisse:

- **/bin** Grundlegende Systemprogramme wie z.B. *cp*, *date*, *ls*, *mkdir*,...; diese Programme sind teilweise auch für den Systemstart wichtig
- **/boot** Boot-Lader und Programme; hier liegt u.a. der Linux-Kernel sowie Parameterdateien
- **/dev** Geräte-Dateien
- **/etc** hier sind rechnerspezifische Konfigurationsdateien aller Art enthalten, z.B. für Netzwerkeinstellungen
- **/home** Home-Verzeichnisse der Benutzer des Systems; jeder Benutzer hat hier ein eigenes Verzeichnis für seine persönlichen Daten, Einstellungen, etc.
- **/lib** wichtige Systembibliotheken und Module (z.B. C-Library *libc*)
- **/opt** Anwendungsprogramme
- **/root** Home-Verzeichnis des Systemadministrators, auch root genannt
- **/sbin** wichtige Programme für die Systemverwaltung (z.B. *fdisk*, *fsck*)
- **/tmp** temporäre Dateien, benutzbar von allen Benutzern
- **/usr** nicht hostspezifische, shareable Daten und Programme
- **/var** variable Daten, z.B. Log-Files, Maildaten, Druckerspooler

Weitere Informationen über den FHS finden Sie unter `http://www.pathname.com/fhs/`

5.2 Dateisysteme

Unter Linux sind eine Vielzahl moderner aber auch ältere Dateisysteme (im Nachfolgenden kurz „DS“ genannt) nutzbar. Die Eigenschaft, auch mehrere verschiedene DS im laufenden Betrieb zu unterstützen, einzubinden (siehe Kapitel 5.6) und wieder auszuhängen, ermöglicht es Linux sehr flexibel, vor allem in Multiboot-Systemen (also Systemen mit mehreren installierten Betriebssystemen), zu agieren.

5.2.1 Betriebsebenen auf der Dienste-Schicht

Dateisysteme muss man im Wettsteinschen Schichtenmodell auf der Dienste-Ebene ansiedeln. Die Dienst-Ebene kann man noch unterteilen in eine *logische* und *reale* Betriebsebene. Die logische Betriebsebene stellt dem Benutzer eine logische Sichtweise von Betriebsmitteln dar. Zu

dieser gehört alles, was nicht direkt einer Hardwarekomponente entspricht; also z.B. eine Datei, Verzeichnis, etc. Die reale Betriebsebene entspricht allen Hardwarekomponenten; es findet ein Mapping (also eine Zuordnung) von logischen auf reale Betriebsmittel statt, d.h. im Dateibaum des Systems entspricht die logische Ebene `/dev/hda` der realen Ebene der ersten Festplatte am ersten IDE-Controller im System.

5.2.2 Dateisysteme

Festplatten können mehrere Partitionen mit DS enthalten. Genaueres zum Erstellen von Partitionen und den notwendigen Befehlen finden Sie in Kapitel 5.7. An dieser Stelle genügt es, zu wissen, dass eine Partition (wenn sie denn genutzt werden soll) ein DS enthalten muss. Andere Datenträger und Wechselmedien wie CD-ROMs, DVDs, USB-Massenspeicher etc. enthalten in der Regel auch DS.

Grundsätzlich muss man die unterschiedlichen DS, die unter Linux nutzbar sind, in verschiedene Kategorien aufteilen, welche Attribute wie Journaling, Einsatzgebiet, ... wiederspiegeln. Die Unterscheidung ‚von welchem Betriebssystem bzw. Hersteller dieses kommt, spielt eher eine untergeordnete Rolle, da Linux, wie bereits erwähnt, flexibel mit verschiedensten DS von verschiedensten Herstellern umgehen kann. Manchmal muss man hierbei allerdings Einschränkungen hinnehmen; so gibt es beispielsweise für NTFS von Microsoft noch keinen verläßlichen Treiber, welcher es ermöglicht auch schreibend auf NTFS zuzugreifen. Dies liegt vor allem an der Lizenzpolitik von Microsoft, welche die Funktionsweise von NTFS nicht offenlegt.

Die im Folgenden aufgezählten Dateisysteme sind lediglich eine Auswahl derer, die unter Linux unterstützt werden:

- **ext2:** das „Standard“ DS von Linux; Dateinamen können bis zu 255 Zeichen lang sein, die maximale Größe für eine Partition liegt bei 2 GB
- **ext3:** basiert auf ext2, wurde jedoch zur Fehlerminimierung und -korrektur um *Journaling*-Funktionalität erweitert
- **FAT:** Standard-DS von MS-DOS und Windows 3.11 - Windows95
- **FAT32:** Standard-DS von Windows98 - WindowsME
- **VFAT:** erweiterte Variante von FAT, Dateinamen bis zu einer Länge von 255 Zeichen mit Unterscheidung von Groß- und Kleinschreibung erlaubt
- **NTFS:** Standard-DS ab Windows NT 4 bis zu aktuellen Windows-Versionen; dieses Dateisystem kann zuverlässig nur Read-Only betrieben werden; Schreibzugriffe sind nur experimentell

und auf eigene Gefahr hin möglich!

- **HPFS:** DS von IBM OS/2
- **ReiserFS:** Journaling-DS von Hans Reiser; hohe Datensicherheit
- **Linux Swap:** DS für die Auslagerungs-Partition (Swap-Partition) von Linux

Wechseldatenträger enthalten andere DS. CD-ROMs, aber auch DVDs, enthalten das ISO-9660-Dateisystem. Dieses Dateisystem ist von der International Standards Organization standardisiert und stellt das „Standard“ Format dar. Weitere Informationen zu DS gibt es unter `http://www.linux-ag.de/linux/LHB/node153.html`.

5.2.3 Inodes

Inodes sind die Verwaltungseinheiten für das Dateisystem eines Linuxsystems. Jedes Objekt im Dateibaum hat ein entsprechendes Inode im Dateisystem, d.h. alle Verzeichnisse, Dateien, ... aus dem Verzeichnisbaum haben einen Verzeichniseintrag im Dateisystem. Inodes enthalten wichtige Informationen über die Objekte:

- den Speicherort des Objekts auf der Festplatte
- Zeitstempel der letzten Änderung des Objekts
- Sicherheitseinstellungen
- Eigentümer
- ...

Multiple Objekte im Dateisystem können sich den selben Inode teilen, dieses Konzept nennt man *Linking*. Mehr dazu finden Sie in Kapitel 5.4.

5.3 Befehle im Zusammenhang mit Dateien und Verzeichnissen

Nachdem im vorangegangenen Kapitel der Dateibaum und Dateisysteme besprochen wurden, fehlen nur noch Befehle, mit denen man navigieren, Dateien lesen und ändern, Verzeichnisse anlegen und löschen kann.

- **cd** (ohne weitere Angaben) wechselt in das Home-Verzeichnis „∼“

- **cd Pfad** wechselt in Verzeichnis *Pfad*, d.h. erklärt den angegebenen Pfad zum aktuellen Verzeichnis
- **ls -l** gibt Verzeichnisinhalt in Liste aus
- **ls -l Dateien** listet Dateien auf
- **ls -l Verzeichnis** listet das Verzeichnis auf
- **ls -lisa** listet Verzeichnis in ausführlicher Form auf
- **pwd** gibt aktuellen absoluten Pfad aus
- **mkdir Verzeichnis** legt Verzeichnis an
- **rmdir Verzeichnis** löscht Verzeichnis; allerdings darf dieses Verzeichnis keine Dateien oder Unterverzeichnisse mehr enthalten

- **cp Quelldatei Zieldatei** legt eine Kopie von Quelldatei unter Zieldatei an
- **cp Quelldatei Verzeichnis** legt eine Kopie von Quelldatei in Verzeichnis an
- **mv Datei-alt Datei-neu** verschiebt Datei-alt in Datei-neu (Umbenennen)
- **mv Datei Verzeichnis** verschiebt Datei-alt in Verzeichnis
- **rm Datei** löscht Datei
- **rm -r Verzeichnis** löscht Verzeichnis unter Rückfrage, ob alle Unterobjekte auch gelöscht werden sollen
- **rm -rf Verzeichnis** löscht Verzeichnis ohne jegliche Rückfrage, ob alle Unterobjekte auch gelöscht werden sollen

- **cat Textdatei** gibt Inhalt der Textdatei aus
- **less Textdatei** gibt Inhalt der Textdatei aus, Blättern und Scrollen ist möglich
- **more Textdatei** gibt Inhalt der Textdatei aus, Blättern und Scrollen ist möglich
- **head Textdatei** gibt die ersten Zeilen der Textdatei aus, z.B. `head -10 brief.txt`
- **tail Textdatei** gibt die letzten Zeilen der Textdatei aus
- **file Datei** zeigt Informationen zum Dateityp an (z.B. ASCII, Binär,...)

5.4 Verweise: Symbolische- und Hard-Links

Verweise (oder auch **Links** genannt) werden dazu benutzt, um einen alternativen Namen zu bestehenden Dateien oder Verzeichnissen zu erstellen.

5.4.1 Symbolischer Link

Symbolische Links funktionieren über Dateisystemgrenzen hinweg (also auch im Netzwerk) und verweisen auf existierende Dateien oder Verzeichnisse. Ein symbolischer Link kann unabhängig von der Zieldatei gelöscht werden. Im Gegensatz zum Hard-Link benötigt ein Symbolischer Link eigenen Speicher im Dateisystem. Konkret handelt es sich um eine kleine Datei welche einen Pointer auf eine andere Datei enthält.

Wenn das Linuxsystem einen solchen Link öffnet, öffnet es die Datei des symbolischen Links und liest den darin enthaltenen Pointer und öffnet daraufhin die eigentliche Datei, auf welche verwiesen wird.

Symbolische Links kann man sich einfach anzeigen lassen mit `ls -l`. Anmerkung: wird eine Datei gelöscht auf welche verwiesen wird, so bleibt der symbolische Link weiterhin existent, zeigt aber ins „nichts“ und wird dann als *stale* bezeichnet.

- `ln -s vorhandenes_objekt symbolischer_name`

5.4.2 Hard-Link

Ein Hardlink ist nicht wirklich ein Verweis, sondern eine Kopie der Verzeichniseinträge (also Inode) einer Datei. Ein Hard-Link hat gegenüber einer Kopie einer Datei den Vorteil, dass der Speicher nur einmal belegt wird und sich beim Ändern einer Instanz auch alle anderen Instanzen ändern.

Wird eine Instanz des Hard-Links, also z.B. die Original-Datei, worauf mehrere Hard-Links verweisen, gelöscht, so zeigen sämtliche verbliebenen Hard-Links immer noch auf dieselbe Inode mit allen Eigenschaften der Datei. Außer dem Namen und dem Speicherort im Dateibaum ist der Hard-Link nicht von dem ursprünglichen Objekt zu unterscheiden. Eine Instanz wird erst gelöscht, wenn alle Instanzen gelöscht sind.

- `ln vorhandene_datei symbolischer_name`

Achtung! Hard-Links haben zwei wesentliche Einschränkungen:

- Ein Hard-Link funktioniert nur im selben Dateisystem! Diese Einschränkung kommt daher, dass eben Inodes geteilt werden zwischen der Original-Datei und dem Hard-Link.
- Hard-Links können nicht auf Verzeichnisse verweisen.

5.4.3 Was ist nun besser?

Symbolische Links werden öfters genutzt als Hard-Links, weil sie vielseitiger und flexibler einsetzbar sind und nur einen sehr kleinen Anteil an Festplattenplatz nutzen.

Wo werden diese Links benutzt?

Wie bereits erklärt wurde, dienen Links dazu einen anderen Namen für eine bestehende Datei oder ein Verzeichnis anzugeben. Ein sinnvoller Anwendungsfall ist die Verwendung von Links im Verzeichnis */etc/rc.d*.

In diesem Verzeichnis sind viele Skripte enthalten, welche Dienste auf dem Linuxsystem starten und stoppen; z.B. einen Webserver (Apache httpd) oder Fileserver (Samba). Im genannten Verzeichnis gibt es noch Unterverzeichnisse *rc0.d*, *rc1.d*, *rc2.d*,..., welche verschiedenen Runleveln des Systems entsprechen.

Jeder Runlevel entspricht einem anderen Zustand des Linuxsystems, d.h. welche Dienste gestartet sind und welche nicht. Also macht es Sinn ein generisches Verzeichnis mit allen Skripten welche laufen können zu definieren; dies ist das Verzeichnis */etc/init.d*. In jedem Verzeichnis *rcx.d* werden nun symbolische Links auf das */etc/init.d*-Verzeichnis gesetzt. Z.B. enthält das Verzeichnis *rc3.d* die folgenden Einträge:

- `S30syslog -> ../init.d/syslog`
- `S40crond -> ../init.d/crond`
- `S85httpd -> ../init.d/httpd`

Man sieht, dass alle Links auf die Skripte im Verzeichnis *../init.d* verweisen. Dies hat eine Menge Vorteile, da so z.B. Änderungen an den Skripten nur an einer Stelle, nämlich in *init.d* gemacht werden müssen und nicht separat in jedem *rcx.d*-Verzeichnis. Weiterhin wird dadurch eine Menge Festplattenplatz eingespart, da ein Link nur einen Bruchteil einer richtigen Kopie des Skripts im jeweiligen Verzeichnis ausmacht. Das Konzept von init.d, rc.d und Runleveln wird im Kapitel 6.6 ausführlicher ausgeführt.

5.5 Zugriffsrechte

Das Konzept der Zugriffsrechte spielt eine sehr große Rolle bei Linux, sowohl die Systemsicherheit betreffend, als auch die Sicherheit jedes einzelnen Benutzers. Datenschutz und Datensicherheit sind somit in jedem Linux-System einfach zu realisieren. In jedem Multiuser-Betriebssystem sind diese Sicherheiten zwingend; jeder Benutzer könnte potentiell wichtige Betriebssystemteile wie Kernel, Konfigurations-Dateien und Programme versehentlich oder absichtlich löschen. Benutzerdaten müssen vor Änderungen durch andere Nutzer geschützt werden und manchmal auch komplett uneinsehbar für andere sein.

5.5.1 Linux Access Control

Die Zugriffsrechte in Linux entsprechen einer Menge von Eigenschaften, welche für jedes Objekt in dessen Inode gespeichert wird. Diese Eigenschaften werden auch *Access Mode* oder kurz *Mode* genannt.

Es gibt drei unterschiedliche Rechte:

- **r - -** **r**ead - Leserecht (Inhalt der Datei einsehen, bzw. Verzeichnisinhalt anzeigen)
- **- w -** **w**rite - Schreibrecht (In Datei schreiben od. ändern, bzw. in Verzeichnis Dateien anlegen oder löschen)
- **- - x** **e**xcecute - Ausführungsrecht (Datei als Programm ausführen, bzw. lesen od. schreiben von Dateien im Verzeichnis)

Man unterscheidet zwischen drei verschiendenen Klassen von Benutzern:

- **rwx - - - - - -** **u**ser der Benutzer welcher die Datei besitzt (owner)
- **- - - rwx - - -** **g**roup die Gruppe welche die Datei besitzt
- **- - - - - - rwx** **o**ther alle anderen Benutzer im System

Es gibt darüberhinaus noch spezielle Zugriffsrechte, welche Benutzerabhängigkeiten (SUID-Bit), Gruppenabhängigkeiten (SGID-Bit) und andere Zugriffe auf Dateien und Verzeichnisse (Sticky-Bit) betreffen; diese werden an dieser Stelle allerdings nicht weiter vertieft.

5.5.2 Setzen und Ändern von Zugriffsrechten

Zugriffsrechte können einfach mit verschiedenen Befehlen gesetzt und geändert werden. Dies kann entweder über Text oder entsprechenden Oktalzahlen geschehen. Der Befehl für das Setzen von Zugriffsrechten ist

- `chmod` *Rechte Objekt*

Mittels Text (Beispiele)

- `chmod g+w` *Objekt* gibt Gruppenangehörigen Schreibrechte
- `chmod o-x` *Objekt* nimmt allen übrigen Nutzern die Ausführungsrechte
- `chmod go=` *Objekt* setzt die Rechte für group und other auf keine Rechte

Dabei gibt es die *Operationen* - (wegnehmen von Rechten), + (hinzufügen) und = (Recht „exakt“ setzen). Alle anderen Buchstaben entsprechen den Benutzergruppen, bzw. den entsprechenden Rechten.

Mittels Oktalzahlen

Das Setzen und Ändern von Zugriffsrechten mittels Oktalzahlen ist etwas schwieriger zu verstehen als mittels Text; erweißt sich jedoch in der alltäglichen Linux-Praxis doch als umso einfacher wenn man das Konzept verstanden hat. Wenn z.B. die Rechte `chmod 754 brief.txt` gesetzt werden bedeuted dies, dass die Datei *brief.txt* vom Benutzer vollständig gelesen, bearbeitet und gelöscht werden kann. Mitglieder der Gruppe des Benutzers dürfen die Datei lesen und ausführen (was bei der Textdatei allerdings keinen Sinn machen würde) und alle Anderen (others) dürften die Datei ebenfalls lesen, also einsehen.

Wie funktioniert das?

Lässt man sich den Verzeichniseintrag von *brief.txt* mittels `ls -l brief.txt` anzeigen erhält man z.B. folgende Ausgabe:

```
-rwxr-xr--   1 alex   users   7 2004-06-06 14:14 brief.txt
```

Betrachtet man die erste kryptische Zeichenfolge, so spiegeln diese Buchstaben die Zugriffsrechte für die verschiedenen Benutzer wieder. Wenn man das erste Bit weglässt (nicht weiter wichtig an dieser Stelle), so haben die folgenden 9 Bits die Bedeutung:

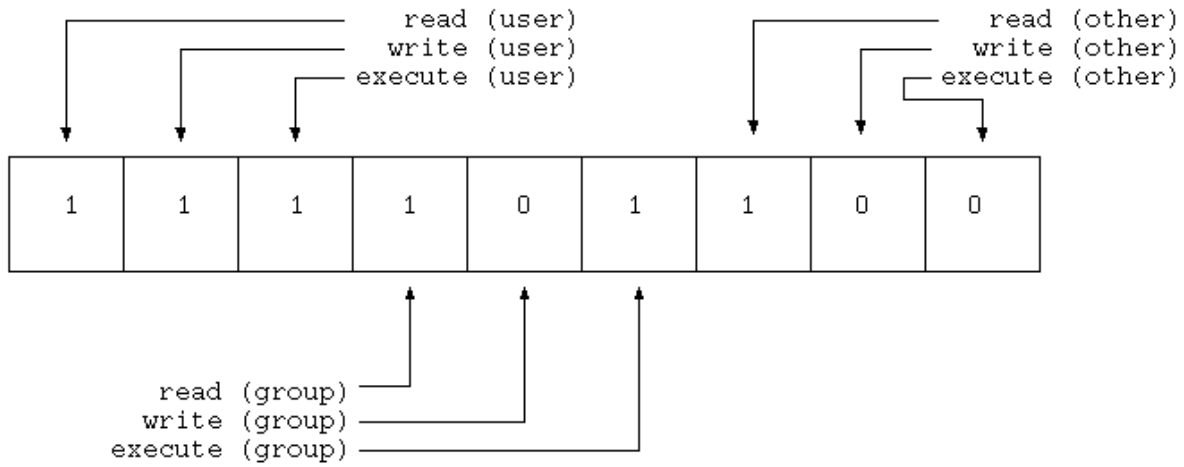

Abbildung 5.3: Mode-Bits einer Datei

In unserem Beispiel sind die jeweiligen Bits für user, group und others auf „1“ gesetzt, wenn diejenigen die Rechte haben sollen.

Jede Dreiergruppe an Bits kann mittels einer Oktalzahl durch Umrechnung Binär → Oktal dargestellt werden. Die entsprechende Umrechnungstabelle sieht dann so aus:

- *Oktalzahl: 0 entsprechende Binärfolge: 000*
- *Oktalzahl: 1 entsprechende Binärfolge: 001*
- *Oktalzahl: 2 entsprechende Binärfolge: 010*
- *Oktalzahl: 3 entsprechende Binärfolge: 011*
- *Oktalzahl: 4 entsprechende Binärfolge: 100*
- *Oktalzahl: 5 entsprechende Binärfolge: 101*
- *Oktalzahl: 6 entsprechende Binärfolge: 110*
- *Oktalzahl: 7 entsprechende Binärfolge: 111*

Anderes Beispiel: möchte man z.B. eine Datei vor Lesen und Ändern durch andere Benutzer schützen, allerdings Gruppenmitglieder das lesen erlauben, darf man nur den enstprechenden Benutzern Rechte dazu geben und es allen anderen verweigern.

Die entsprechenden Mode-Bits müssen also für user auf „111“ für group auf „100“ und für other auf „000“ gesetzt werden, was der Binärfolge 111 100 000 entspricht. Ausgedrückt als Oktalzahlen ist dies 7 4 0, also muss man für die Datei den Befehl `chmod 740 datei` verwenden.

5.6 Einbinden von Dateisystemen (Mounten)

Unter dem Einbinden (Einhängen) von Dateisystemen, oder auch *mounten* genannt, versteht man das Einbinden von Partitionen von Festplatten, CD-ROMs, DVDs oder anderen Wechselmedien wie USB-Sticks. Dieses Konzept wird durchgehend in allen Unix und Linux-Systemen verfolgt.

5.6.1 Mounten

Um eine Partition zu mounten muss man zuerst einen *Mountpoint*, also eine Art Ankerpunkt definieren, an dem im Dateibaum das Dateisystem gemountet werden soll. Für diesen Ankerpunkt wird ein normales Verzeichnis verwendet, d.h. man legt ein neues Verzeichnis an (z.B. */mnt/cdrom*) und kann nun das CD-ROM mit dem Befehl

- `mount /dev/cdrom /mnt/cdrom`

in dieses Verzeichnis mounten. */dev/cdrom* entspricht hier der Quelle, d.h. das Gerät, bzw. die Partition auf welchem das Dateisystem sich befindet. Natürlich kann dieser Gerätename auf anderen Systeme komplett anders lauten, z.B. */dev/scd0* für ein SCSI-CD-ROM Laufwerk. I.d.R. werden die DS-Typen automatisch erkannt. Nun kann man die gemountete CD-ROM wie ein ganz normales Verzeichnis benutzen, d.h. Dateien lesen, kopieren, etc.

Natürlich kann man das Dateisystem auch wieder *unmounten*, also aushängen:

- `umount /mnt/cdrom`

Dabei ist zu beachten, dass im entsprechenden Befehl tatsächlich das „n“ fehlt :o)

5.6.2 fstab und mtab

In jedem Linux-System gibt es die Dateien */etc/fstab* und */etc/mtab*.

- **fstab:** In dieser Datei werden sämtliche Einträge gemacht, welche Dateisysteme zum Systemstart des Linux-Systems gemounted werden sollen
- **mtab:** Hier werden alle aktuell gemounteten Dateisysteme und Mountpoints aufgelistet

Damit ein Linux-System überhaupt lauffähig ist, wird als erstes das Root-Dateisystem „/“ gemountet - danach dann, während des Starts, alle weiteren Mountpoints aus der *fstab*. Hier einige typische Zeilen aus der *fstab*:

```
/dev/hda3   /   ext3   defaults 1 1
/dev/hda1   /windows/C   ntfs   ro,auto,users 0 0
/dev/hda2   swap   swap   pri=42 0 0
/dev/hdb1   /windows/D   vfat   rw,auto,user,exec 0 0
```

Diese Einträge mounten die verschiedenen Festplattenpartitionen von */dev/hda* und */dev/hdb* auf die jeweiligen Mountpoints im Dateibaum, so z.B. auch die Swap-Partition. Hierbei wird für jedes DS der DS-Typ angegeben (z.B. ntfs); man kann stattdessen auch auto angeben, wobei beim mounten versucht wird automatisch den Typ festzustellen.

Exemplarisch soll die Zeile für das mounten der ntfs-Partition von Windows genauer betrachtet werden:

- die Partition */dev/hda1* wird auf den Mountpoint */windows/C* mit Dateisystem-Typ *ntfs* gemountet
- hierbei werden weitere Optionen angegeben: *ro* = readonly, also kann das Dateisystem nur gelesen werden; *user* = jeder User darf einen (u)mount auf das Dateisystem ausführen.
- die zwei Ziffern am Ende der Zeile haben eher eine geringfügigere Bedeutung; die erste Ziffer wird bei Systemdatenträgern auf „1“ gesetzt, sonst auf „0“. Die zweite Ziffer gibt an, in welcher Reihenfolge die Partitionen bei einem check überprüft werden sollen; für Systemdatenträger ist dies „1“, für sonstige Linux-Partitionen „2“ und alle anderen bekommen „0“.

5.6.3 fstab Dateisystemtypen

In der fstab kann man jederzeit weitere Einträge vornehmen um neue Dateisysteme zu mounten. Falls man neue Einträge gemacht hat kann man sämtliche Mountpoints mittels `mount -a` mounten.

Folgende Dateisystemtypen (auszugsweise) können in der fstab verwendet werden:

- **auto** Dateisystem automatisch bestimmen
- **ext2**, **ext3** Linux ext2 bzw. ext3
- **iso9660** CD-ROMs, DVDs
- **nfs** Network File System
- **ntfs** WinNT NTFS

- **reiserfs** ReiserFS
- **smbfs** SambaFS (Windows-Netzwerkverzeichnis)
- **swap** Swap-Partition
- **udf** Universal Disk Format (DVDs, CD-RWs)
- **usbdevfs** USB-Geräte (*/proc/bus/usb*)
- **vfat** Windows-9x Vfat-FS

5.7 Partitionierung

Die Festplatten-Partitionierung - vielleicht sollte man gleich zu Beginn erwähnen, dass man mit dem Partitionieren viel „kaputt“ machen kann auf einer Festplatte. Man sollte sich jederzeit seines Handelns bewusst sein und nicht voreilig Partitionen anlegen, löschen oder verändern, denn so hat man schnell ein lauffähiges Linux- oder Windows-System unbrauchbar gemacht.

5.7.1 Organisation

Partitionieren = Organisieren

In vielen PCs gibt es, die Dateisystemorganisation betrachtet, nur eine Partition, worauf alle Daten gespeichert werden; also Betriebssystemdaten, Benutzerdaten und Anderes. In Windows-Systemen ist dies typischerweise „das Laufwerk C:“. Wird in einem Windows-System nun eine zweite Festplatte installiert und formatiert, so werden Laufwerksbuchstaben wie D:, E:, usw. hinzugefügt - der Benutzer muss nun wissen auf welchem Laufwerk er seine Benutzerdaten speichert. Diese Einfachheit kann für viele Benutzer von Vorteil sein, andere Nutzer bevorzugen es jedoch, dass man dies flexibler handhaben kann.

In Linux-Systemen ist dies sehr viel flexibler handhabbar.

Ein Beispiel: In einem Server gibt es für Benutzerdaten (also die Home-Verzeichnisse der Benutzer) eine Festplatte mit 10 GB Speicherplatz, welche nach */home* gemounted ist. Diese Festplatte wird nun langsam voll aufgrund exzessiven Nutzerverhaltens; es wird bald mehr Speicherplatz benötigt. Die einfachste Möglichkeit ist es für den Server-Administrator sämtliche Daten von der 10GB-Festplatte auf eine neue Festplatte mit mehr Kapazität zu kopieren und die neue Platte einfach wieder unter */home* zu mounten. Natürlich muss vorher die alte 10GB-Platte geunmounted werden. Wenn der Benutzer nun beim nächsten Zugriff auf sein Home-Verzeichnis ganz normal

weiterarbeitet, hat er von dieser Erweiterung der Kapazität nichts mitbekommen, da er wie gewohnt unter seinem Home-Verzeichnis weiterarbeiten kann.

Festplatten: Geräte-Namen

Grundlegend muss man wissen wie Linux Festplatten und Partitionen auf Festplatten „benennt", dies folgt einem einfachen Schema.

Für IDE-Festplatten gilt die folgende Reihenfolge:

- */dev/hda* - primary master IDE (oftmals die Festplatte)
- */dev/hdb* - primary slave IDE
- */dev/hdc* - secondary master IDE (oftmals auch CD-ROM Laufwerk)
- */dev/hdd* - secondary slave IDE

Für SCSI-Devices ist dies ähnlich:

- */dev/sda* - erstes SCSI-Device
- */dev/sdb* - zweites SCSI-Device
- */dev/sdc* - drittes SCSI-Device ... (usw.)

Festplatten: Partitionen

Die Partitions-Nummerierung folgt nun einem ähnlich einfachen Schema, für die 4 primären Partitionen (*primary partitions*) gilt (hier beispielsweise für */dev/hda*)

- */dev/hda1*
- */dev/hda2*
- */dev/hda3*
- */dev/hda4*

Neben den primären Partitionen gibt es noch erweiterte Partitionen (*extended partitions*). Eine erweiterte Partition ist eine Variante einer primären Partition, kann aber kein eigenes Dateisystem beinhalten. Anstatt dessen enthält eine erweiterte Partition logische Partitionen (*logical partitions*)

welche die Nummern 5-16 besitzen. Insgesamt kann es entweder 4 primäre Partitionen geben, oder eben anstatt einer der primären Partitionen eine erweiterte Partition welche wiederum bis zu 12 logische Partitionen enthält. Mit diesem Schema können also pro Festplatte ein Maximum von 15 Partitionen adressiert werden (3 primäre + 12 logische Partitionen). Um diesen relativ einfachen Zusammenhang besser zu verstehen, folgend ein Beispiel (Abbildung 5.4):

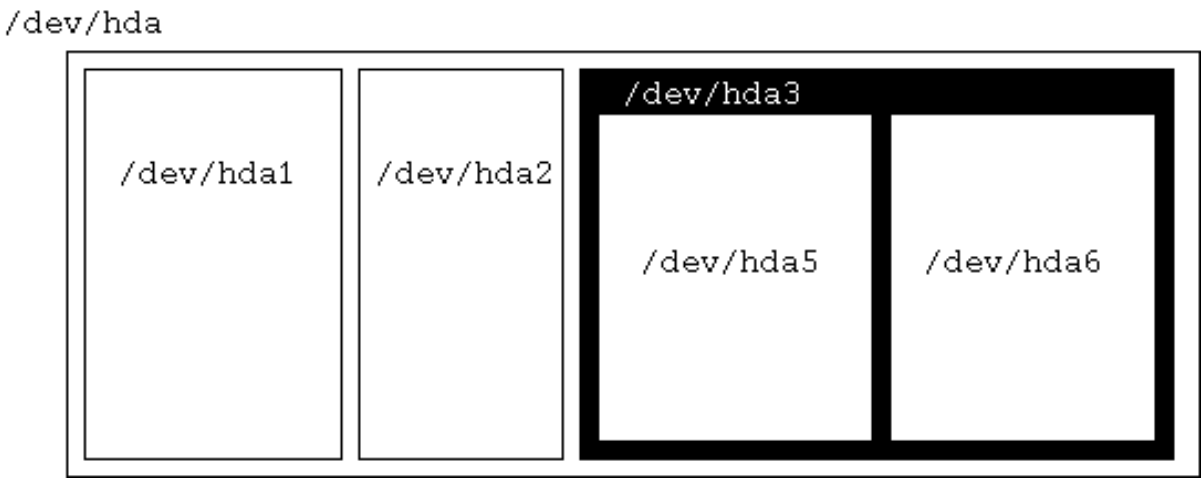

Abbildung 5.4: Partitionen einer Festplatte (/dev/hda)

Exemplarisch wurde die *primary master* Festplatte verwendet; diese enthält hier zwei primäre Partitionen */dev/hda1* und */dev/hda2*. Bei der Partition */dev/hda3* handelt es sich um eine erweiterte Partition, welche die logischen Partitionen */dev/hda5* und */dev/hda6* enthält. So sind hier vier verschiedene Partitionen zur Datenspeicherung nutzbar.

5.7.2 Befehle

fdisk

Um Partitionen zu erstellen und zu verwalten gibt es das Programm *fdisk*. Der Aufruf des Programms erfolgt immer unter Angabe der zu bearbeitenden Festplatte, also z.B. `fdisk /dev/hda`. Wenn man danach im Menü „m“ aufruft, erhält man eine Hilfeseite mit allen möglichen Optionen:

Dabei haben die am häufigsten gebrauchten Menüpunkte folgende Bedeutungen:

- **a :** Flag setzen ob Partition bootbar sein soll oder nicht
- **d :** Partition löschen; die Eingabe der Partitionsnummer ist erforderlich
- **l :** Liste aller möglichen Partitionstypen anzeigen (Tabelle)

```
gonzo:~ # fdisk /dev/hda

The number of cylinders for this disk is set to 2584.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): m
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos compatibility flag
   d   delete a partition
   l   list known partition types
   m   print this menu
   n   add a new partition
   o   create a new empty DOS partition table
   p   print the partition table
   q   quit without saving changes
   s   create a new empty Sun disklabel
   t   change a partition's system id
   u   change display/entry units
   v   verify the partition table
   w   write table to disk and exit
   x   extra functionality (experts only)
```

Abbildung 5.5: Partitionieren mit fdisk - Hilfetext

- **m :** das Hilfemenü aufrufen (Abbildung 5.5)
- **n :** Partition hinzufügen; es wird nach Typ (primary, extended, logical) und entsprechender Nummer gefragt; automatisch wird der Startzylinder auf der Festplatte für die Partition eingetragen und für die Partitionsgröße kann man den Endzylinder angeben, oder auch eine Größe wie z.B. „+1500M“ für 1500MB (1,5GB)
- **p :** Partitionstabelle ausgeben, welche beim Start von fdisk in den Hauptspeicher eingelesen wurde
- **q :** Beenden von fdisk **OHNE** Änderungen an der Partitionstabelle
- **t :** Typ einer Partition ändern (z.B. von ext2 auf ext3)
- **w :** Beenden von fdisk **MIT** Änderungen an der Partitionstabelle; diese wird komplett neu auf die Festplatte geschrieben.

Partitionen anzeigen

Die Partitionen der Festplatte kann man sich dann ganz einfach per Befehl „**p**“ ausgeben lassen:

```
Command (m for help): p

Disk /dev/hda: 20.0 GB, 20003880960 bytes
240 heads, 63 sectors/track, 2584 cylinders
Units = cylinders of 15120 * 512 = 7741440 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1               1        1500    11339968+   7  HPFS/NTFS
/dev/hda2            1501        1570      529200   82  Linux swap
/dev/hda3   *        1571        2472     6819120   83  Linux
```

Abbildung 5.6: fdisk: Anzeige der Partitionen

In dieser Konfiguration hat die Festplatte eine NTFS-Partition */dev/hda1*, eine Swap-Partition */dev/hda2*, eine Partition mit einem Linux-System */dev/hda2*. Weiterhin kann man noch sehen, dass für jede Partition eine *ID* angegeben wird. Diese IDs geben den Partitionstyp an; man kann sich alle möglichen Typen mit dem Befehl „**l**“ anzeigen lassen:

```
Command (m for help): l

 0  Empty           1c  Hidden Win95 FA 70  DiskSecure Mult bb  Boot Wizard hid
 1  FAT12           1e  Hidden Win95 FA 75  PC/IX           be  Solaris boot
 2  XENIX root      24  NEC DOS         80  Old Minix       c1  DRDOS/sec (FAT-
 3  XENIX usr       39  Plan 9          81  Minix / old Lin c4  DRDOS/sec (FAT-
 4  FAT16 <32M      3c  PartitionMagic  82  Linux swap      c6  DRDOS/sec (FAT-
 5  Extended        40  Venix 80286     83  Linux           c7  Syrinx
 6  FAT16           41  PPC PReP Boot   84  OS/2 hidden C:  da  Non-FS data
 7  HPFS/NTFS       42  SFS             85  Linux extended  db  CP/M / CTOS / .
 8  AIX             4d  QNX4.x          86  NTFS volume set de  Dell Utility
 9  AIX bootable    4e  QNX4.x 2nd part 87  NTFS volume set df  BootIt
 a  OS/2 Boot Manag 4f  QNX4.x 3rd part 8e  Linux LVM       e1  DOS access
 b  Win95 FAT32     50  OnTrack DM      93  Amoeba          e3  DOS R/O
 c  Win95 FAT32 (LB 51  OnTrack DM6 Aux 94  Amoeba BBT      e4  SpeedStor
 e  Win95 FAT16 (LB 52  CP/M            9f  BSD/OS          eb  BeOS fs
 f  Win95 Ext'd (LB 53  OnTrack DM6 Aux a0  IBM Thinkpad hi ee  EFI GPT
10  OPUS            54  OnTrackDM6      a5  FreeBSD         ef  EFI (FAT-12/16/
11  Hidden FAT12    55  EZ-Drive        a6  OpenBSD         f0  Linux/PA-RISC b
12  Compaq diagnost 56  Golden Bow      a7  NeXTSTEP        f1  SpeedStor
14  Hidden FAT16 <3 5c  Priam Edisk     a8  Darwin UFS      f4  SpeedStor
16  Hidden FAT16    61  SpeedStor       a9  NetBSD          f2  DOS secondary
17  Hidden HPFS/NTF 63  GNU HURD or Sys ab  Darwin boot     fd  Linux raid auto
18  AST SmartSleep  64  Novell Netware  b7  BSDI fs         fe  LANstep
1b  Hidden Win95 FA 65  Novell Netware  b8  BSDI swap       ff  BBT
```

Abbildung 5.7: fdisk: Partitionstypen

Partitionen anlegen

Neue Partitionen anzulegen und zu formatieren ist sehr einfach zu bewerkstelligen. Als sinnvolles Beispiel wollen wir eine „Transferpartition“ einrichten. Die Transferpartition soll zum Datenaustausch zwischen Windows und Linux verwendet werden, da auf der einen Seite Linux zuverlässig nur lesend auf NTFS-Partitionen zugreifen kann. Der Schreibzugriff auf NTFS-Partitionen ist noch nicht zuverlässig möglich, da Microsoft aus Lizenzgründen das Dateiformat nicht offenlegt. Auf der anderen Seite kann Windows standardmäßig nicht auf Linux-Dateisysteme zugreifen; hier bedarf es Zusatzsoftware. Die einfachste Lösung ist es also auf der Festplatte eine Art Transfer-Partition einzurichten, welche von beiden Betriebssystemen gelesen und geschrieben werden kann. Zum einfachen Datenaustausch eignet sich hierfür am besten eine FAT16/FAT32 Partition.

Für das folgende Beispiel nehmen wir an, dass sich am Ende der Festplatte noch ca. 820MB Platz befinden, um die Partition erstellen zu können. Alle Schritte können in Bild 5.8 (im Screenshot) nachvollzogen werden:

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Selected partition 4
First cylinder (2473-2584, default 2473):
Using default value 2473
Last cylinder or +size or +sizeM or +sizeK (2473-2584, default 2584): +800M

Command (m for help): t
Partition number (1-4): 4
Hex code (type L to list codes): c
Changed system type of partition 4 to c (Win95 FAT32 (LBA))

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
```

Abbildung 5.8: fdisk: neue Partition anlegen

Die Schritte gliedern sich also wie folgt:

- **Command: n :** mit diesem Command wird das Untermenü für das Anlegen einer Partition aufgerufen

  - zuerst muss man angeben, ob man eine extended oder primary Partition anlegen will; wir legen eine primäre Partition an
  - die nächste Frage ist der Startzylinder; automatisch wird der Erste freie verfügbare auf der Festplatte gewählt (2473), welchen man mit einem einfachen Bestätigen durch Enter auswählt
  - nun fehlt nur noch die Größe der Partition: dies kann man entweder angeben, indem man den letzten Zylinder angibt oder die größe Absolut angibt; mögliche Eingaben sind hier z.B. +820M (für Megabyte) oder +82000K (für Kilobyte)

- **Command: t :** mit diesem Commando gibt man den Partitionstyp an; da wir eine FAT32-Partition wollen schauen wir in der Tabelle aus Bild 5.7.3 nach und wählen Typ c. Man kann sich hier auch per L auch nochmal diese Tabelle anzeigen lassen.
- **Command: w :** hat man alles erledigt, so muss man die geänderte Partitionstabelle noch auf die Festplatte schreiben. Dies sollte man allerdings nur dann tun, wenn man sich absolut sicher ist :o) - danach wird automatisch fdisk beendet.

Natürlich kann man (bevor man alles auf die Festplatte schreibt) sich nochmal den Inhalt per **p** anzeigen lassen und erhält folgende Anzeige:

```
Command (m for help): p

Disk /dev/hda: 20.0 GB, 20003880960 bytes
240 heads, 63 sectors/track, 2584 cylinders
Units = cylinders of 15120 * 512 = 7741440 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1               1        1500    11339968+   7  HPFS/NTFS
/dev/hda2            1501        1570      529200   82  Linux swap
/dev/hda3   *        1571        2472     6819120   83  Linux
/dev/hda4            2473        2584      846720    c  Win95 FAT32 (LBA)
```

Abbildung 5.9: fdisk: neue Partitionstabelle

Formatieren der neuen Partition

Natürlich muss die neue Partition auch noch für die Verwendung formatiert werden. Dies geschieht mit dem Befehl **mkfs**. Normalerweise werden Linux-Partitionen mit dem ext2-Dateisystem formatiert, jedoch können auch beliebige andere DS erzeugt werden. Tatsächlich ist *mkfs* ein Frontend für Programme wie *mkfs.ext2*, *mkfs.ext3*, *mkfs.msdos*, ... Diese Programme werden auch nochmal spezifisch verlinkt durch z.B. *mke2fs* und *mkdosfs*. mkfs bietet ein Front-End für all diese Programme.

Wir legen nun ein neues Dateisystem für unsere Transferpartition an mit dem Befehl

- `mkfs -t msdos /dev/hda4 -cv`

Hierbei bedeuten die Optionen *-t msdos*, dass eben ein MSDOS-Dateisystem zur Grundlage liegt, */dev/hda4* ist die Partition welche formatiert werden soll, und *-cv* bedeutet, dass nach „Bad Blocks“ gesucht werden soll (check) und alles detailliert ausgegeben werden soll (verbose). Man erhält Beispielsweise die folgende Ausgabe:

```
gonzo:~ # mkfs -t msdos /dev/hda4 -cv
mkfs.msdos 2.8 (28 Feb 2001)
/dev/hda4 has 240 heads and 63 sectors per track,
logical sector size is 512,
using 0xf8 media descriptor, with 1693440 sectors;
file system has 2 16-bit FATs and 32 sectors per cluster.
FAT size is 207 sectors, and provides 52906 clusters.
Root directory contains 512 slots.
Volume ID is 40c849e3, no volume label.
Searching for bad blocks 47168... 94400... 142208... 187584... 228928... 2760963
24352... 370816... 417792... 464688... 511344... b559024... 602880... 635648...
68544... 700800... 733824... 766208... 799360... 833088...
```

Abbildung 5.10: mkfs: formatieren

Natürlich kann die Partition entsprechend Kapitel 5.6 gemounted werden. Für weitere Optionen und Informationen zu fdisk, mkfs, ... stehen Ihnen jederzeit die Man-Pages der jeweiligen Programme zur Verfügung :o)

5.8 Dateisystem-Integrität

Unter Dateisystemintegrität versteht man die Tatsache, dass die Datenträger keine Fehler im Dateisystem oder Dateibaum enthalten und dass diese auch nicht demnächst „vollaufen“, d.h. immer noch Speicherplatz für das System wichtige Prozesse enthalten. Diese Fehler können auftreten wenn z.B. ein Stromausfall auftritt während der PC läuft, oder eben auch ganz einfach (technische) Ursachen haben, wie z.B. dass die maximal Anzahl der Inodes im Dateisystem überschritten wurde und so keine neuen Objekte mehr angelegt werden können. Für das Sicherstellen dieser Integrität gibt es eine Vielzahl von Tools, von denen nun eine Reihe vorgestellt werden.

5.8.1 df - disk free

Der Befehl *df* wird dazu benutzt, den freien Speicherplatz auf der Festplatte und die Inodes zu überwachen. Wie bereits weiter oben auf dieser Seite angesprochen, kann einem Dateisystem die

Anzahl der Inodes „ausgehen“, da Inodes bereits beim erstellen des Dateisystems mittels *mkfs* erstellt werden. Inodes können z.B. dann ausgehen, wenn im Dateisystem sehr viele kleine Dateien enthalten sind, die jeweils eine eigene Inode benötigen.

Mögliche Aufrufe von df:

- **df** *Optionen Verzeichnisse* allgemeine Syntax des Befehls
- **df -h** gibt freien Speicherplatz auf allen gemounteten Dateisysteme aus; *-h* ist die Option für *human readable*, d.h. dass die Ausgabe in einem einfach lesbaren Format mit Mega- und Gigabyte-Angaben erfolgt. Die Platzausnutzung wird auch Prozentual ausgegeben.
- **df .** hiermit kann man einfach feststellen, auf welcher Partition man sich gerade befindet

5.8.2 du - disk usage

du dient der Anzeige des Platzverbrauchs eines oder mehrerer Verzeichnisse.

Mögliche Aufrufe von du:

- **du** *Optionen Verzeichnisse* allgemeine Syntax des Befehls
- **du -sh /home/alex** gibt die Speicherplatznutzung des Verzeichnisses */home/alex* aus, auch hier wieder im human-readable format. Option *-s*, also *mit* Unterverzeichnissen
- **du -Ssh /etc** gibt Speicherplatznutzung des Verzeichnisses */etc* aus, allerdings -S ohne Unterverzeichnisse
- **du -csh /home/*** gibt Summary (*-c*) für alle Unterverzeichnisse in */home* aus

5.8.3 fsck - file system check

Mittels *fsck* können bereits aufgetretene und vorhandene Inkonsistenzen im Dateisystem aufgespürt und beseitigt werden. Dabei ist fsck ein Frontend für Programme wie *fsck.ext2* oder *fsck.ext3* (Vergleich Kapitel 5.7 *mkfs*) Eine wichtige Information welche auf eine Partition geschrieben wird ist die *Superblock*-Information. Der Superblock ist Block 1 der Partition und enthält wichtige Informationen. Wenn der Superblock durch auftretende Fehler zerstört ist, kann fsck diese wieder herstellen. Dazu wird der Superblock bereits beim Anlegen des Datesystems redundant im Dateisystem abgelegt, i.d.R. alle 8192 Blocks; also ist die erste Kopie bei 8193, die zweite bei 16385, usw zu finden.

Mögliche Aufrufe von fsck:

- **fsck** *[Optionen] [-t Typ] [DS-Optionen] Dateisysteme* allgemeine Syntax des Befehls
- **fsck /dev/hda3** checkt das Dateisystem auf */dev/hda3* ; wenn Dateisystem „clean“ aussieht wird check abgebrochen
- **fsck -f /dev/hda3** checkt das Dateisystem auf */dev/hda3 -f* „force“, d.h. auch wenn DS „clean“ aussieht wird check gemacht
- **fsck -vf /dev/hda3** dasselbe wie `fsck -f /dev/hda3`, jedoch mit erweiterter Anzeige
- **fsck -y /dev/hda3** Nachfragen werden automatische alle mit „yes“ beantwortet

Die am häufigsten gebrauchten Optionen für fsck:

- **-A** lässt Check für alle in */etc/fstab* aufgeführten DS durchführen; diese Option ist primär für den Bootvorgang des Linuxsystems gedacht.
- **-N** fsck *nicht* ausführen, nur anzeigen was getan werden würde
- **-t** *Typ* Typ des Dateisystems angeben; der hier angegebene Typ entscheidet, welcher *fsck.x* aufgerufen wird.

Die am häufigsten gebrauchten Optionen für e2fsck (ext2):

- **-b** *superblock* benutze eine alternative Kopie des Superblocks, automatisch werden die Backups des Superblocks an den Stellen 8193, ... genommen
- **-c** nach „Bad Blocks“ durchsuchen
- **-f** Check forcieren, auch wenn DS „clean“ aussieht
- **-p** Reparieren ohne Rückfragen
- **-y** auf alle Rückfragen mit „yes“ antworten, dadurch kann *e2fsck* nicht-interaktiv benutzt werden.

6 Programme/Prozesse

In diesem Kapitel geht es um den Zusammenhang zwischen Programmen, Prozessen und weiteren automatisierten Diensten (sog. Daemons) von Linux. Genauso wie die Administration des Dateisystems eine grundlegende Tätigkeit eines Administrators ist, ist das „Pflegen“ und Überwachen von Prozessen eine grundlegende Aufgabe. Meistens werden Programme und Prozesse vom Kernel und vom System automatisiert gehandelt, d.h. überwacht, in Ihrem Zustand geändert oder gelöscht. Manchmal passiert es eben auch, dass Prozesse „sterben“ oder anderweitig außer Kontrolle geraten und dann harmlose oder weniger harmlose Effekt auslösen. Genau an dieser Stelle setzt dieses Kapitel an, nämlich vom Grundlegenden Verständnis von Prozessen aus bis hin zur Überwachung und Kontrolle.

Die folgenden Punkte sind der Inhalt dieses Kapitels; die hier genannten Befehle stellen nur eine Übersicht über die wichtigsten Befehle der nachfolgenden Kapitel dar. In den jeweiligen Kapiteln werden auch weitere wichtige Befehle aufgezeigt.

- 6.1 Basics - Konzept von Programmen, Prozessen, Tasks
- 6.2 Prozesszustände
- 6.3 Pipes, Ein-/Ausgabeumleitung
- 6.4 Prozesskontrolle (Befehle: *ps, pstree, top, kill*)
- 6.5 Daemons
- 6.6 init.d - rc.d

6.1 Programme = Prozesse ?

6.1.1 Programme und Prozesse

Programme sind ausführbare Dateien welche z.B. auf der Fesplatte gespeichert sind.

Prozesse dagegen sind **Instanzen** eines Programms, d.h. ein Programm wird geladen und im Hauptspeicher des PCs ausgeführt (es befindet sich in Ausführung).

Prozesse werden oftmals auch als **Tasks** bezeichnet. Dies kann man durchaus als Synonym für Prozesse benutzen. Allerdings sei an dieser Stelle gesagt, dass Linux ein *Multitasking* Betriebssystem ist, in welchem Prozesse parallel laufen können. Man darf dies allerdings nicht mit *Multiprocessing* verwechseln: dies bedeuted, dass bei entsprechender Hardware und angepasstem Linux-Kernel ein Linux-System mehrere Prozesse parallel auf verschiedenen Prozessoren verarbeiten kann. Genaueres hierzu finden Sie im Kapitel 6.2.

6.1.2 Prozesse

Jeder Prozess, der im Linux-System läuft, wird vom Kernel überwacht. So ist z.B. auch die Shell, in welche man Kommandos eintippt, ein Prozess und ebenso der Webbrowser, auf welchem Sie gerade diese Zeilen lesen oder der X-Server, durch welchen der Webbrowser dargestellt wird.

Prozesse können auch **Subprozesse** beinhalten, d.h. Prozesse, die von einem übergeordneten Prozess abhängig sind und von diesem „Vater-Prozess“ gesteuert werden oder eben unabhängig von diesem laufen. Zu jeder Zeit können mehrere Zehn bis Hundert Prozesse gleichzeitig reibungslos im System laufen.

Damit all dies gemanagt werden kann, haben Prozesse mehrere Attribute und verfolgen verschiedene Konzepte:

- **Lebenszeit** (Lifetime) : jeder Prozess „lebt“, wenn er sich in Ausführung befindet; so werden Programme wie z.B. ls nur für kurze Zeit ausgeführt, ein Ergebnis wird generiert und ausgegeben und danach wird der Prozess terminiert. Wenn ein Prozess terminiert, so stirbt er; deshalb gibt es zum Beenden von Prozessen auch das Programm *kill*.
- **Prozess ID** (Process ID, PID) : für jeden Prozess eindeutige Integer-Nummer.
- **Benutzer ID und Gruppen ID** (User ID UID, Group ID GID) : der Benutzer, der den Prozess startet, gibt automatisch dem Prozess seine UID und GID mit; mit diesen Rechten agiert der Prozess dann auch und darf nur auf die Ihm erlaubten Ressourcen zurückgreifen.
- **Vaterprozess** (Parent process) : der erste im System gestartete Prozess ist *init* (siehe Kapitel 6.6); von Ihm abgeleitet starten alle anderen Prozesse.
- **Vaterprozess ID** (Parent PID) : die ID des Prozesses, von dem der aktuelle Prozess abhängig ist.

- **Umgebung** (Environment) : jedem Prozess ist eine Liste von Variablen und Werten zugeordnet; dies Variablen werden auch *Umgebungsvariablen* (environment variables) genannt.
- **Arbeitsverzeichnis** (Current working directory) : jedem Prozess wird ein Arbeitsverzeichnis mitgegeben, in dem er standardmäßig Dateien schreiben und lesen kann, außer man gibt ein anderes Verzeichnis explizit an.

6.2 Prozesszustände

Damit Linux auch wirklich multiple Prozesse scheinbar gleichzeitig bearbeiten kann, bedient es sich dem Modell des *Multitaskings*. Vorneweg möchten wir allerdings an dieser Stelle darauf hinweisen, dass die folgende Betrachtung zu Prozess-Multitasking und deren Zustände sehr vereinfacht betrachtet wird.

6.2.1 Multitasking

Das Umschalten zwischen mehreren gleichzeitig laufenden Prozessen wird von einem *Scheduler* erledigt, welcher fest in den Kernel integriert ist. Dieser Scheduler teilt Prozessen Rechenzeit auf dem/den real vorhandenen Prozessor(en) zu. Dabei wird dynamisch reguliert: hat ein Prozess gerade nichts zu tun, so wird diesem Rechenzeit entzogen und einem anderen Prozess zugeteilt. Ein anderes Modell ist z.B. das Zuteilen einer Maximal-Zeit für jeden Prozess. Wenn ein Prozess dann eine bestimmte Rechenzeit überschreitet, so wird diesem Prozessorzeit entzogen und eben einem Anderen zugewiesen.

Dabei ist es wichtig, wie ein Prozess umgeschaltet wird: in der Regel wird ein Prozess angehalten, bzw. weggeschaltet oder zurückgestellt und ein anderer Prozess kommt an die Reihe. Ist irgendwann der ursprüngliche Prozess wieder an der Reihe, so wird dieser fortgesetzt an der Stelle wo er unterbrochen wurde. Natürlich ist es wichtig, dass sämtliche Speicherinhalte welche dem Prozess zugeordnet waren, zuerst gesichert wurden damit diese wieder genauso hergestellt werden können.

Es gibt zwei Arten von Multitasking:

- **kooperatives Umschalten (cooperative multitasking) :** beim kooperativen Multitasking wird, vereinfacht gesagt, die Kontrolle des Umschaltens, also wann, wie und wohin, fast komplett den Anwendungsprogrammen überlassen. Oftmals kommt es hier eben zu Timingproblemen oder Überschreibungen im Speicher, bzw. Prozesse „hängen“ einfach, so dass die Betriebssysteme, welche diese Art des Multitasking einsetzen, oftmals hängenbleiben oder abstürzen (Beispiel: Atari ST, Risc OS, Windows95, ...).

- **preemptives Umschalten (cooperative multitasking) :** auch Umschalten mit Verdrängung genannt; die vollständige Kontrolle über die Prozesszeiten werden vom Kern, bzw. Prozessor übernommen. Prozesse können also unfreiwillig stillgelegt, bzw. vom Prozessor verdrängt werden. Jedes moderne Betriebssystem wie Linux, Unix, Windows XP,... benutzt preemptives Multitasking.

6.2.2 Prozesszustände

Damit das Umschalten auch funktioniert und die Prozesse angehalten und wieder fortgesetzt werden können vom Scheduler, muss es definierte Prozesszustände geben. Diese elementaren Zustände stellen durch definierte Übergänge sicher, dass hier kein Prozess einfach so z.B. zum „Geist“ werden kann:

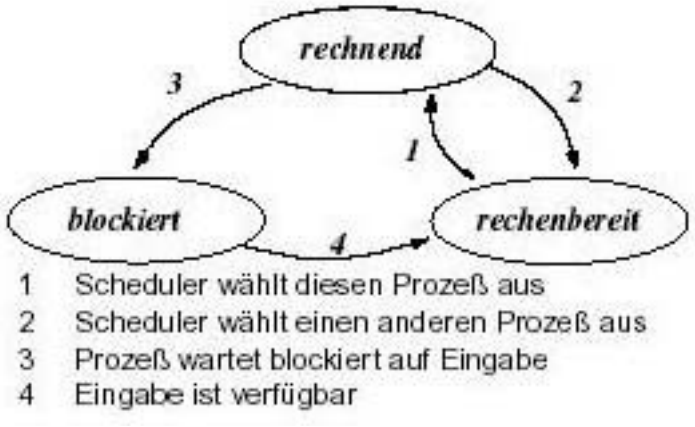

Abbildung 6.1: Elementare Prozesszustände [20]

Die zentralen Zustände sind *Rechnend, Bereit und Blockiert (Wartend)*. Der Scheduler muss nun auswählen, welcher Prozess als nächster zum Rechenzeit auf dem Prozessor zugewiesen bekommt: dies ist der rechenbereite Prozess, da nur dieser in Frage kommt. Der Blockierte Prozess wartet z.B. auf eine Eingabe, also macht es keinen Sinn, dass dieser Rechenzeit zugeteilt bekommt. Hat dieser allerdings später dann eine Eingabe bekommen, so wird dieser in den Zustand rechenbereit überführt und bekommt als Nächster Rechenzeit zugewiesen. Die typische Auswahlmenge bei Prozessen eines Prozessors sind also rechenbereite Prozesse sowie der auf dem jeweiligen Prozessor rechnende Prozess.

6.2.3 Schnittstellen von Prozessen

Prozesse haben Schnittstellen, welche Ihnen erlauben Steuersignale zu empfangen. Jeder laufende Prozess hört ständig, ob solche Signale an ihn gesendet werden. Durch Interprozess-Kommunikation werden diese Signale übertragen und sind genauer betrachtet jeweils nur eine Integer-Zahl, welche über eine definierte Schnittstelle übertragen wird. Genauso ist auch definiert, wie die Prozesse auf diese Signale zu reagieren haben. Wenn man z.B. Ctrl-C während eines laufenden Programms drückt, so wird die Nachticht „INTERRUPT“ an das Programm gesendet, welches dem Prozess sagt, dass dieser stoppen soll.

Es sind ca. 30 Signale in Linux definiert. Jedes Signal hat einen Namen und eine Nummer, wobei der Name nur zur besseren Unterscheidung für uns Menschen dient, da ja bei der eigentlichen Kommunikation nur eine Integer-Zahl übermittelt wird. Viele von diesen Signalen werden vom Kernel benutzt und einige sind auch für den normalen Benutzer nötig. Einige häufig benutzte Signale sind:

- **SIGHUP (Integer 1) : Hang Up;** dieses Signal wird automatisch gesendet wenn man sich ausloggt von einer Konsole oder eine Modemeinwahl unterbricht
- **SIGINT (Integer 2) : Interrupt;** Stop der Ausführung, z.B. durch Ctrl-C
- **SIGKILL (Integer 9) : Kill;** sofortiges unkontrolliertes Stoppen des Prozesses; dieses Signal kann nicht ignoriert werden vom Prozess; z.B. durch den Befehl *kill*
- **SIGTERM (Integer 15) : Terminate;** wenn möglich „sanft“, d.h. der Prozess wird gefragt ob er sich selbst kontrolliert beenden kann
- **SIGTSTP (Integer 20) : Stop** der Ausführung, Bereitmachen für späteres Fortführen, also so eine Art „Pause“, z.B. durch Ctrl-Z

Die komplette Liste der Signale finden Sie unter `man 7 signal`.

6.3 Streams, Umleitungen, Pipes

6.3.1 Unix Streams

Ein Grundlegendes Konzept von Unix/ Linux wurde bereits in Kapitel 5.1 besprochen: *alles im System ist eine Datei*. Alles, das sind u.a. Festplattenlaufwerke und deren Partitionen; CD/DVD-Laufwerke; Tapes; Terminals (z.B. die Konsole); Serielle, Parallele und andere Schnittstellen; Maus,

Tastatur, uvm. All diese Elemente werden als Datei gemappt - und diese Eigenart macht das Handling und die Interaktion zwischen diesen Elementen sehr flexibel, da Schnittstellen standardisiert sind.

Eine Festplatte beispielsweise, angesprochen als Blockdevice über */dev/hda*, wird durch eben diese Gerätedatei repräsentiert. Das gegebene Interface lässt es zu, dass mittels einfachen Systembefehlen wie z.B. dd von dieser Festplatte ein komplettes Backup in eine Datei, also ein sogenanntes *Image* gemacht werden kann. Ein anderes Beispiel wäre z.B. ein Programm, welches von der Konsolendatei (z.B. */dev/ttyx* eine Eingabe erwartet, d.h. wenn jemand auf dieser Konsole tippt, werden die Zeichen sofort zum Programm umgeleitet. Dieses Mapping übernimmt der Kernel.

Unter *Unix Streams* versteht man die Fähigkeit von Objekten, ihre *Ein- und Ausgaben* auf standardisierte Art und Weise zu verknüpfen. Diese Fähigkeit wird von der Shell zur Verfügung gestellt. Wird also ein Programm gestartet, so hat es automatisch drei Deskriptoren (*file descriptors*); in Abbildung 6.2 sieht man eine Übersicht über den Aufbau der Standard-I/Os.

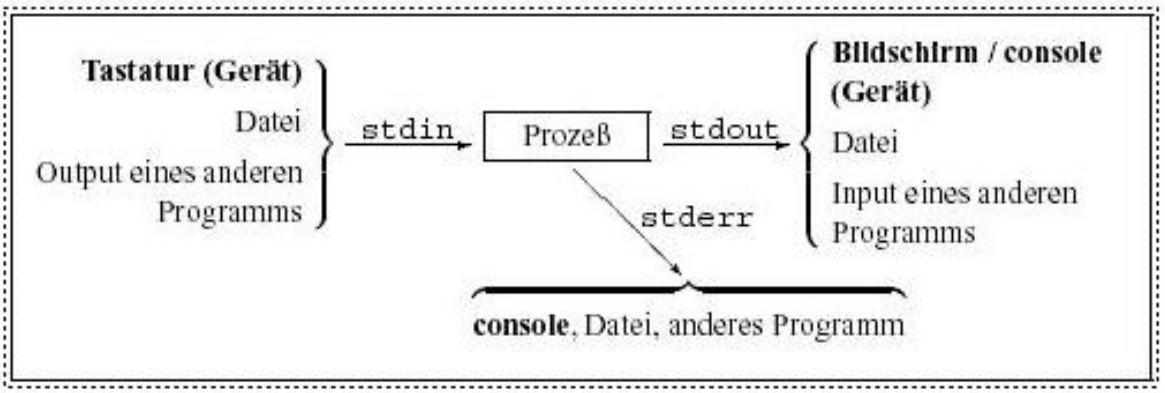

Abbildung 6.2: Standard-I/O eines Programms [20]

- **stdin - Standard Input (Standardeingabe) :** dieser Deskriptor ist ein Text Eingabe-Stream; standardmäßig wird dieser aufs Keyboard gemappt; wenn man also eine Eingabe macht wird dies von Programmen, welche eine Eingabe erwarten, ausgewertet. Bei vielen Programmen ist stdin auch ein oder mehrere Kommandozeilenargument(e), welche Dateien, Verzeichnissen, oder anderem entsprechen.
- **stdout - Standard Output (Standardausgabe) :** dieser Deskriptor ist ein Text Ausgabe-Stream; standardmäßig ist dieser an das Terminal gebunden, d.h. man sieht sämtliche Ausgaben im Terminal. Diese Ausgabe kann auch umgeleitet werden, z.B. in eine Datei.
- **stderr - Standard Error (Standardfehlerausgabe) :** dieser Deskriptor ist ein Text Ausgabe-Stream; er wird exklusiv nur für Fehlermeldungen und anderen eher unwichtigen Informationen verwendet; standardmäßig ist dieser ebenso dem Terminal zugeordnet, d.h. man sieht

sämtliche Ausgaben sowohl von stdout als auch von stderr auf dem Terminal; dies kann oftmals verwirrend sein.

Die Trennung von stout und stderr macht Sinn, da es oftmals Hilfreich ist Fehlermeldungen separat zu handeln.

6.3.2 Redirection - Umleitung

Die Standard-IOs kann man beliebig flexibel umleiten - dies geht sehr einfach über die Konsole mittels den reservierten Zeichen „<“ und „>“. Die Umleitung kann sowohl zu Dateien hin als auch von Dateien weg stattfinden. Oftmals macht es Sinn große Datenbestände, z.B. Listen, welche durch ein Programm generiert werden, in eine Datei umzuleiten, mit welcher man dann später einfacher arbeiten kann, um die Inhalte auszuwerten.

- `kommando < datei`

  Standardeingabe aus Datei

- `kommando > datei`

  Standardausgabe in Datei umleiten (Datei wird überschrieben)

- `kommando 1> datei`

  Standardausgabe in Datei umleiten (Datei wird überschrieben)

- `kommando 2> datei`

  Standardfehlerausgabe in Datei umleiten

- `echo "Dies ist das Ende" >> datei`

  schreibt „Dies ist das Ende“ an Ende der Datei

- `kommando > datei 2>&1`

  leitet stdout und stderr in datei um

- `kommando > datei1 2> datei2`

  leitet stdout in datei1 und stderr in datei2 um

6.3.3 Pipes

Aus der Sichtweise eines Programms macht es keinen Unterschied, ob die Standardeingabe von der Konsole oder aus einer Textdatei kommt. Genauso ist es einem Programm relativ egal, ob

es seine Standardaushabe auf eine Konsole oder in eine Datei schreibt. Als Erweiterung dieser Idee ist es möglich, die Standardausgabe eines Programms mit der Standardeingabe eines anderen Programms zu verknüpfen und ist somit ist ein „Spezialfall“ einer Redirection. Erreicht wird dies mittels einer *Pipe* „|“. Dabei kann man auch mehr als zwei Programme miteinander „pipen“, in diesem Fall wird der Textfluss auch *Pipeline* genannt. Die Fehlerausgabe mittels stderr kann auch für alle Programme in Dateien umgeleitet werden, siehe „Redirection“.

- `kommando1 | kommando2`

  leitet stdout von kommando1 in stdin von kommando2 um

- `kommando1 < eingabendatei 2> fehlerdatei1 | kommando 2> fehlerdatei2 | kommando3`

  diese etwas komplexere Reihenfolge zeigt dass man recht einfach kommandos pipen und die Fehlerausgabe in Dateien umleiten kann; als Status dieser Pipeline wird der Status des letzten Programms angenommen

Einige Beispiele für Pipes:

- `ls | wc -l`

  zählt die Anzahl der Objekte in einem Verzeichnis und gibt die Zahl aus

- `ls -alF | more`

  zeigt den Verzeichnisinhalt Seitenweise an

- `grep "70771" kundendatensatz* | less`

  zeigt eine scrollbare Liste aller Zeilen der Dateien, an welche mit „70771“ beginnen und deren Dateinamen mit kundendatensatz* beginnen

6.4 Prozesskontrolle

Prozesskontrolle beinhaltet zum einen das Überwachen von Prozessen und deren Zustände, und zum anderen das Senden von Signalen an diese Prozesse, um deren Zustand zu beinflussen, bzw. einen Prozess komplett zu beenden.

6.4.1 Überwachung von Prozessen

Zur Überwachung dienen drei Kommandos, welche auf jedem Unix-/ Linuxsystem vorhanden sind, und, welche nun im Einzelnen vorgestellt werden.

Befehl: ps

Dieser Befehl erzeugt eine Momentaufnahme aller aktiven Prozesse im System und listet diese in einer Liste auf, so z.B. in Abbildung 6.3 für den Benutzer (`ps x`).

```
alex@gonzo:~> ps x
  PID TTY      STAT   TIME COMMAND
 1340 ?        S      0:00 /bin/sh /usr/X11R6/bin/kde
 1359 ?        S      0:00 /usr/bin/ssh-agent -s
 1376 ?        S      0:00 kdeinit: Running...
 1379 ?        S      0:00 kdeinit: dcopserver --nosid
 1382 ?        S      0:00 kdeinit: klauncher
 1385 ?        S      0:01 kdeinit: kded
 1393 ?        S      0:00 kdeinit: klaptopdaemon
 1395 ?        S      0:01 /opt/kde3/bin/artsd -F 10 -S 4096 -s 5 -m artsmessage -l 3 -f
 1402 ?        S      0:00 kdeinit: knotify
 1405 ?        S      0:00 kwrapper ksmserver
 1407 ?        S      0:00 kdeinit: ksmserver
 1408 ?        S      0:02 kdeinit: kwin -session 117f000002000107369039800000085810000_1086966018_780380
 1410 ?        S      0:03 kdeinit: kdesktop
 1412 ?        S      0:03 kdeinit: kicker
 1414 ?        R      0:00 kdeinit: kio_file file /tmp/ksocket-alex/klauncheriRaJLa.slave-socket /tmp/ksocket-alex/
 1417 ?        S      0:00 kdeinit: klipper
 1420 ?        S      0:00 kdeinit: kwrited
 1422 ?        S      0:01 susewatcher -caption SuSE Updater -icon kinternet.png -miniicon kinternet.png --quiet
 1425 ?        R      0:00 kdeinit: kmix -session 117f000002000107369041400000085810009_1086966015_413216
 1428 ?        S      0:00 kdeinit: konqueror --preload
 1430 ?        S      0:00 kalarmd --login
 1438 ?        R      0:03 kdeinit: konsole
 1439 pts/1    S      0:00 /bin/bash
 1475 pts/1    S      0:58 kile 06programme_prozesse_3pipes.xml
 1478 pts/2    S      0:00 /bin/bash
 1487 ?        S      0:00 /bin/sh /home/alex/firefox/run-mozilla.sh /home/alex/firefox/firefox-bin
 1493 ?        S      0:18 /home/alex/firefox/firefox-bin
 1494 ?        S      0:00 /home/alex/firefox/firefox-bin
 1495 ?        S      0:00 /home/alex/firefox/firefox-bin
 1496 ?        S      0:00 /home/alex/firefox/firefox-bin
 1609 ?        S      0:00 /usr/bin/ssh-agent -s
 1668 pts/3    R      0:00 /bin/bash
 1717 pts/3    R      0:00 ps x
```

Abbildung 6.3: Ausgabe des Befehls *ps* zur Anzeige aller laufenden Prozesse des Users

Die am häufigsten gebrauchten Optionen von *ps*:

- **-a :** zeige *alle* im System laufenden Prozesse an, auch die anderer User; normalerweise werden nur die vom jeweiligen User angezeigt
- **-f :** „forest“ Modus; Subprozesse werden in einer baumartigen Struktur an die Vaterprozesse angehängt
- **-l :** Langformat; zeigt Priorität, Vater PID und andere Informationen an
- **-u :** Userformat; zeigt Benutzernamen und Startzeit der Prozesse an
- **-w :** „Wide“ Format; sehr nützlich für die -f Option da die Zeilen am Ende nicht abgeschnitten werden
- **-x :** Prozesse anzeigen, welche von keinem Terminal kontrolliert werden (z.B. Systemprozesse wie Daemons)

Einige Beispiele zum Aufruf von *ps*:

- `ps`

  zeigt eine einfache Liste von Prozessen an, welche dem aufrufenden User gehören

- `ps aux`

  zeigt eine Umfassende Liste aller Prozesse im System an, inkl. Startzeit und Usernamen

- `ps auxf`

  dasselbe, allerdings werden Subprozesse in einer Art Baumstruktur dargestellt

Befehl: pstree

Dieser Befehl erzeugt eine Momentaufnahme aller aktiven Prozesse im System und listet diese in einer Baumstrukturartigen Liste auf und entspricht so dem Aufruf von `ps -f`. In Abbildung 6.4 sehen wir beispielhaft einen solchen Baum (`pstree -p`).

Die am häufigsten gebrauchten Optionen von *pstree*:

- **-a :** zeige Kommandozeilenargumente an, mit welchen Prozesse aufgerufen wurden
- **-c :** Zusammenfalten von identischen Unterbäumen verhindern
- **-h :** hervorheben der Hostorie des momentanen Prozesses
- **-n :** sortieren nach PID
- **-p :** Anzeige aller Prozesse mit PID

Befehl: top

Ähnlich zum Kommando *ps* erzeugt *top* eine ständig aktualisierende Ansicht aller laufenden Prozesse. Dies kann zur forlaufenden Kontrolle eines Prozesses sehr hilfreich sein. Beim standardmäßigen Aufruf von *top* wird die Anzeige alle 5 Sekunden aktualisiert und die am rechenintensivsten Prozesse werden zuerst aufgeführt. Weiterhin werden eine Reihe Zusatzinformationen wie CPU-Status, Speicherinformationen, uvm. angezeigt (Abbildung 6.5).

Die am häufigsten gebrauchten Kommandozeilen-Optionen von *top*:

- **-d :** Delay; Anzahl der Sekunden zwischen den Updates

- **-i :** Idle-Prozesse ignorieren, also nur Prozesse anzeigen welche aktiv sind
- **-n :** Anzahl der Iterationen wie oft Anzeige upgedated wird anstatt unendlich zu laufen, danach beenden von top
- **-s :** „secure“ Modus; manche Optionen im interaktiven Modus können gefährlich sein; diese Option deaktiviert diese

Einige Optionen im interaktiven Modus von *top*:

- **Ctrl-L :** Bildschirm neu aufbauen
- **h :** Hilfe anzeigen
- **k :** „kill“ eines Prozesses, also beenden; PID muss nach Druck auf k eingegeben werden
- **n :** Anzahl der zu zeigenden Prozessen ändern; es muss eine Integer-Zahl eingegeben werden, Standard ist „0“
- **q :** Beenden des Programms
- **s :** Delay zwischen den Updates; es können auch Bruchwerte angegeben werden, z.B. 0.5

6.4.2 Beeinflussung

Beeinflussen von Prozessen Zum Beeinflussen von Prozessen stehen mehrere Befehle zur Verfügung. Weiterhin kann über die Shell auch das Verhalten von Prozessen beeinflusst werden.

Befehl: kill

Dieser Befehl wird benutzt, um einen Prozess zu beenden. Genauer gesagt sendet der Befehl *kill* ein gewünschtes Signal an einen Prozess, um diesen entweder auf sanfte oder harte Weise zu beenden (vgl. Kapitel 6.2). Das Signal kann entweder mit Namen angegeben werden (z.B. `kill -SIGTERM`, oder mittels des entsprechenden Integer-Wertes (z.B. `kill -s 15`). Mittels `kill -l` kann man sich eine Liste ausgeben lassen mit allen möglichen Signalen. Wenn ein kill-Befehl an einen Prozess geschickt wird, so werden i.d.R. auch seine Kindprozesse beendet.

Im Standardaufruf mit `kill` *PID*, also unter Angabe der Prozess ID, wird versucht den Prozess auf eine „sanfte“ Art und Weise zu beenden, und zwar mittels SIGTERM. Hiermit wird dem Prozess Zeit gegeben, sich kontrolliert zu beenden, d.h. wenn z.B. noch etwas in eine Datei geschrieben werden muss, kann er dies tun bevor er sich beendet.

Manchmal ist ein Prozess jedoch so blockiert, dass er nicht mehr auf das Signal antworten kann. Dann hilft nur noch den Prozess regelrecht „abzuschießen“ mittels **kill -9** (oder auch `kill -KILL`). Hierbei wird der Prozess allerdings mit sofortiger Wirkung beendet; sollten also noch Informationen in eine Datei geschrieben werden, so sind diese unwiederbringlich verloren.

Manchmal wird ein Prozess in *ps* oder *top* auch als *Zombie* aufgeführt und kann selbst mittels `kill -9` nicht beendet werden. Diese Prozesse befinden sich gerade im Status des Beendens, sind aber irgendwo steckengeblieben. Genauso wie man es aus Hollywoodfilmen kennt, so können Zombies nicht mehr ge*kill*t werden, da sie bereits tot sind.

Hier ein Beispiel zum Aufruf von *kill*; sämtliche folgende Aufrufe zum sofortigen Beenden vom Prozess 4711 können synonym verwendet werden:

- `kill -9 4711`
- `kill -KILL 4711`
- `kill -SIGKILL 4711`
- `kill -sigkill 4711`
- `kill -s 9 4711`
- `kill -s SIGKILL 4711`

Jobkontrolle per Shell

Linux bietet die Möglichkeit, Programme, also deren laufende Prozesse, über die Shell zu kontrollieren. Wenn ein Programm, z.B. `ls` gestartet wird, so läuft es im Vordergrund (*foreground* **fg**) und nimmt Eingaben über das Terminal an und gibt Ausgaben aufs Terminal aus. Wenn es im Hintergrund (*background* **bg**) ausgeführt wird, dann kann man es nicht direkt beeinflussen, sondern nur über das Senden von Signalen (z.B. mittels `kill`). Wird ein Programm im Hintergrund ausgeführt, spricht man auch von einem *Job*.

Was für einen Sinn macht das? Nun, es gibt eine vielzahl von Programmen, welche nicht direkt eine Eingabe oder Ausgabe mittels Terminal erfordern. Startet man z.B. den Webbrowser Mozilla Firefox will man ja diesen Browser direkt nutzen und nicht über die Shell. Die Shell ist allerdings blockiert wenn man das Programm ausführt, außer man sagt explizit, dass das Programm im Hintergrund laufen soll. Dies geschieht ganz einfach per Anhängen des **&**-Zeichens an ein Programm, also z.B. `firefox &`. Man erhält eine Ausgabe in der Form *[1] 4711*, wobei die Zahl in Klammern die Jobnummer angibt, gefolgt von der zugeteilten PID. Startet man ein Programm ohne „&“,

so kann man es nachträglich auch noch in den Hintergrund versetzen, indem man es mit **Ctrl-Z** stoppt und dann per Eingabe von **bg** in den Hintergrund versetzt.

Mittels der Eingabe von jobs erhält man eine Liste aller im Hintergrund laufenden Prozesse in der Form *[1]- Running gimp &*, wobei immer die Jobnummer in [ ] Klammern steht, gefolgt vom Programmname. Will man einen Job wieder in den Vordergrund holen, so genügt es dies mittels Eingabe von `fg` *jobnummer* zu tun.

6.5 Daemons

Die meisten Systemdienste welche in einem Linuxsystem laufen werden über *Daemons* realisiert. Daemons sind Programme welche beim Systemstart über verschiedene Skripte gestartet werden und dann im Hintergrund laufen um Dienste anzubieten.

Viele Netzwerk-Services sind typische Dameons welche auf einem spezifischen Port „lauschen“, ob Anfragen hereinkommen; so lauscht z.B. der sshd (SSH Daemon) auf Port 22. Jede Anfrage wird vom Daemon gehandelt, angefangen vom Einloggen bis hin zum Anbieten verschiedenster Dienste. Weitere typisch Daemons welche laufen können sind z.B.:

- **dhcpd** Daemon um DHCP-Anfragen zu beantworten
- **httpd** Webserver, z.B. Apache
- **inetd** der „Internet-Superdaemon“ (siehe weiter unten)
- **telnetd** Telnet-Serverdienst
- **pppd** PPP-Dienst (Point-to-Point Protocol)
- **lpd** Printerdienst

6.5.1 inetd

Damit nicht unnötig viel Daemons im System laufen welche nicht ständig benötigt werden, gibt es das Konzept des **inetd**, des *Internet-Superdaemons*. Anstatt dass viele einzelne Daemons ständig laufen für Netzwerkdienste, läuft nur der inetd als einziger Daemeon welcher auf alle benötigten Portnummern lauscht (z.B. für SSH, Telnet, FTP,...). Kommt nun eine eingehende Anfrage herein (z.B. für SSH), so wird der entsprechende sshd gestartet und die Anfrage an diesen weitergereicht. So wird sichergestellt dass nur die Ressourcen auch wirklich verbraucht werden welche benötigt werden. Jedoch macht dies nur Sinn bei sporadisch gebrauchten Diensten, da jedesmal

der entsprechende Daemon neu gestartet werden muss. Will man einen Apache Webserver betreiben welcher stark frequentiert ist, so ist es besser den httpd separat laufen zu lassen damit dieser Verbindungsanfragen selbst entgegennehmen kann.

Für die Konfiguration des inetd gibt es die Datei **/etc/inetd.conf**; dort findet das Mapping statt, welcher Daemon für welche Verbindungsanfrage benutzt werden soll. Als Beispiel soll die Zeile für einen FTP-Server genügen:

- # ftp stream tcp nowait root /usr/sbin/tcpd vsftpd

Hier kann man deutlich sehen, dass für eingehende Verbindungsanfragen für FTP der tcpd gestartet wird, mit einem Handle für vsftpd, welcher einen FTP-Server darstellt.

6.6 init.d

6.6.1 intit

Init ist der Vaterprozess *aller* Prozesse. Seine Hauptaufgabe ist es, Prozesse zu generieren, welche in dem Skript */etc/inittab* hinterlegt sind. Hier werden z.B. die Konsolen gestartet, auf welchen man sich dann als Benutzer einloggen kann, aber auch viele andere autonome Prozesse, die für den Betrieb des Systems notwendig sind. Wird `init` als Befehl benutzt, so kann man mit diesem den sogenannten *Runlevel* ändern, d.h. Signale an den initd schicken damit dieser den Runlevel ändert.

6.6.2 Runlevels

Jedes Linux- und viele Unix-Systeme verfolgen das Konzept der Runlevels. Ein Runlevel ist eine Softwarekonfiguration eines Systems ,welche eine bestimmte Gruppe von Prozessen umfasst, die gestartet werden sollen. Damit kann man bestimmen, dass z.B. für einen Runlevel „Webserver“ sämtliche für das System wichtige Dienste und der Webserver-Daemon gestartet werden sollen. Genauso kann man einen Runlevel „Notfall“ definieren, welcher nur eine bestimmte Anzahl wichtiger Dienste beinhaltet die für den Notfallbetrieb wichtig sind.

Die Prozesse sind definiert in der */etc/inittab*; genauer gesagt ist hier ein Verweis für jeden Runlevel auf das Verzeichnis */etc/init.d*, bzw. */etc/rc.d*. Es gibt acht definierte Runlevel: 0-6 und S (oder s). Allerdings muss man beachten dass die Inhalte dieser Runlevel sich von System zu System deutlich unterscheiden können:

- **Runlevel 0 :** Halt des Systems, d.h. Shutdown (Reserviert)
- **Runlevel 1,s,S :** Single-User Mode; hier wird nur das nötigste gestartet, so z.B. keine Netzwerkdienste, File-Sharing,... (Reserviert)
- **Runlevel 2 :** Multiuser ohne NFS File-Sharing
- **Runlevel 3 :** Multiuser inkl. aller wichtigen Netzwerk, Filesharing, etc.-Dienste
- **Runlevel 4 :** wird i.d.R. nicht benutzt
- **Runlevel 5 :** Multiuser, voller Betrieb inklusive grafischer Oberfläche (z.B. KDE); Standard-Runlevel der meissten aktuellen Distributionen
- **Runlevel 6 :** Reboot des Systems (Reserviert)

Die Runlevel 0, 1 und 6 sind also reserviert; theoretisch können auch die Runlevel 7-9 benutzt werden; diese sind aber nicht wirklich dokumentiert. Die Runlevel S und s werden nicht direkt benutzt und sind an dieser Stelle uninteressant fürs weitere Verständnis.

6.6.3 Struktur von rc.d

Die Runlevel selbst sind eher gegenstandslos, da sie auf den ersten Blick „nur“ auf die Verzeichnisstruktur */etc/rc.d* verweisen. Es ist eigentlich der Init-Prozess, welcher aufgrund der Runlevel-Spezifikation Systemeinstellungen ändert und Prozesse startet, welche eben in den Verzeichnissen von *rc.d* vorhanden sind.

Im Verzeichnis */etc/init.d* befinden sich Skripte für System-Services wie z.B. httpd, ftpd, sshd,... . Mittels diesen Skripten können die entsprechenden Dienste gestartet werden, meistens kennen diese Skripte Argumente wie „start“, „stop“, „restart“ oder „status“. Man kann also z.B. den Webserver starten mittels `httpd start`.

Im Verzeichnis */etc/rc.d* befindet sich für jeden Runlevel ein eigenes Unterverzeichnis, also */etc/rc.d/rc0.d* bis *rc6.d*. Diese Verzeichnisse beinhalten Symbolische Links auf die Skripte in */etc/init.d*, und zwar in der Form *[K|S][nn][init.d_name]*, also z.B.:

- *K07alsasound -> ../alsasound** oder auch
- *S14sshd -> ../../init.d/sshd**

Die Symbolischen Links haben eine Menge Vorteile: ändert sich an einem init-Skript etwas, so muss es nur in der Datei im */etc/init.d*-Verzeichnis geändert werden. Sonst müsste man in jedem

Runlevel-Verzeichnis das jeweilige Skript ändern. Weiterhin wird so Speicherplatz eingespart, da die Skripte sonst mehrmals in den Runlevel-Verzeichnissen liegen würden; symbolische Links dagegen benötigen nur einen Bruchteil dieses Platzes. Dabei bedeuten die Buchstaben und Zahlen nach einer Konvention:

- **[K|S] :** Präfixe für *kill* oder *start*; da in jedem Runlevel bestimmte Prozesse laufen oder eben nicht, müssen eben eventuell vorhandene beendet werden und andere laufen.
- **[nn] :** Sequenznummer, diese Integerzahl gibt an in welcher Reihenfolge die Dienste beendet oder gestartet werden, wobei die kleinste Nummer zuerst ausgeführt wird.
- **[init.d_name] :** Der Name des zu beendenden oder startenden Skripts; dieser Name wird allerdings nicht von init ausgewertet, sondern dient der besseren Übersichtlichkeit für uns Menschen.

Wenn nun also z.B. der Runlevel 5 gestartet wird, so werden zuerst alle Dienste mit „K“ beendet und alle Dienste mit „S“ gestartet. Nachdem alle Skripte nacheinander erfolgreich (oder nicht) gestartet worden sind wird eine Meldung vom System ausgegeben: „Runlevel 5 has been reached“.

```
alex@gonzo:~> pstree
init-+-atd
     |-bdflush
     |-cardmgr
     |-cron
     |-cupsd
     |-dhcpcd
     |-kalarmd
     |-kapmd
     |-kdeinit-+-artsd
     |         |-3*[kdeinit]
     |         |-kdeinit-+-bash---kile---bash
     |         |         `-bash-+-gimp---script-fu
     |         |                `-pstree
     |         |-ksnapshot
     |         `-run-mozilla.sh---firefox-bin---firefox-bin---2*[firefox-bin]
     |-11*[kdeinit]
     |-kdm-+-X
     |     `-kdm---kde---kwrapper
     |-keventd
     |-khubd
     |-kinoded
     |-kjournald
     |-klogd
     |-ksoftirqd_CPU0
     |-kswapd
     |-kupdated
     |-lvm-mpd
     |-master-+-pickup
     |        `-qmgr
     |-mdrecoveryd
     |-6*[mingetty]
     |-nscd---nscd---5*[nscd]
     |-portmap
     |-resmgrd
     |-2*[ssh-agent]
     |-sshd
     |-susewatcher
     `-syslogd
```

Abbildung 6.4: Ausgabe des Befehls *pstree* zur Anzeige aller laufenden Prozesse des Users in Baumstruktur

```
top - 20:33:30 up  1:34,  5 users,  load average: 0.52, 0.19, 0.11
Tasks:  76 total,   2 running,  74 sleeping,   0 stopped,   0 zombie
Cpu(s):   1.9% user,   2.9% system,   0.0% nice,  95.2% idle
Mem:    385424k total,   234296k used,   151128k free,    15768k buffers
Swap:   529192k total,        0k used,   529192k free,   101300k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  Command
 1871 alex      15   0   896  896  708 R  3.8  0.2   0:04.13 top
 1312 root      15   0 32440  22m 4524 S  1.0  6.0   1:34.88 X
    1 root      15   0   244  244  208 S  0.0  0.1   0:04.23 init
    2 root      15   0     0    0    0 S  0.0  0.0   0:00.50 keventd
    3 root      15   0     0    0    0 S  0.0  0.0   0:00.01 kapmd
    4 root      34  19     0    0    0 S  0.0  0.0   0:00.01 ksoftirqd_CPU0
    5 root      25   0     0    0    0 S  0.0  0.0   0:00.00 kswapd
    6 root      25   0     0    0    0 S  0.0  0.0   0:00.00 bdflush
    7 root      15   0     0    0    0 S  0.0  0.0   0:00.00 kupdated
    8 root      25   0     0    0    0 S  0.0  0.0   0:00.00 kinoded
   10 root      25   0     0    0    0 S  0.0  0.0   0:00.00 mdrecoveryd
   14 root      15   0     0    0    0 S  0.0  0.0   0:00.08 kjournald
  104 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 lvm-mpd
  563 root      15   0   592  592  496 S  0.0  0.2   0:00.02 syslogd
  566 root      15   0  1356 1356  428 S  0.0  0.4   0:00.19 klogd
  618 root      22   0     0    0    0 S  0.0  0.0   0:00.00 khubd
  741 root      24   0   680  680  500 S  0.0  0.2   0:00.01 cardmgr
  768 root      15   0   648  648  560 S  0.0  0.2   0:00.00 resmgrd
  775 bin       15   0   384  384  316 S  0.0  0.1   0:00.01 portmap
  908 root      15   0  2960 2960 1312 S  0.0  0.8   0:00.50 cupsd
  974 root      17   0  1504 1504 1244 S  0.0  0.4   0:00.14 sshd
 1117 root      15   0  1252 1252 1000 S  0.0  0.3   0:00.02 master
 1137 postfix   15   0  1200 1200  972 S  0.0  0.3   0:00.03 pickup
 1138 postfix   15   0  1228 1228  992 S  0.0  0.3   0:00.01 qmgr
 1155 at        15   0   564  564  500 S  0.0  0.1   0:00.00 atd
 1179 root      15   0   652  652  576 S  0.0  0.2   0:00.00 cron
```

Abbildung 6.5: Ausgabe des Befehls *top* zur Anzeige aller laufenden Prozesse (Ausschnitt)

7 Shell

Die Shell - das Grundlegende Interface für jeden Benutzer eines Computersystems, besonders bei Linux und Unix. Bei aktuellen Distributionen wird die **bash** (Bourne Again SHell) als Standard-Shell für jeden Benutzer eingerichtet.

Natürlich gibt es auch noch weitere Shells wie z.B. die **csh** (C shell), die **tcsh** (eine Variante der csh), oder die „original“ Shell **sh** von Steve Bourne. Jedoch werden wir uns hier in diesem Kurs auf die bash konzentrieren.

Die folgenden Punkte sind der Inhalt dieses Kapitels; die hier genannten Befehle stellen nur eine Übersicht über die wichtigsten Befehle der nachfolgenden Kapitel dar. In den jeweiligen Kapiteln werden auch weitere wichtige Befehle aufgezeigt.

- 7.1 Kommandozeile: Grundlagen, Editor, Tabulator-Taste, Bash-History, Reservierte Zeichen, Quotierung, Erweiterungen, Wildcards
- 7.2 Kommandos: wichtige Shell Kommandos, &&, Substitution, Arithmetische Erweiterung, Aliase
- 7.3 Variablen: Auswahl wichtiger Variablen, Setzen v. Variablen, Export, Path
- 7.4 Shellprogrammierung: Grundlagen, Beispiele

7.1 Kommandozeile

7.1.1 Bash Basics

Die Bash ist ein mächtiges, interaktives Werkzeug - es gibt fast nichts, was man nicht mit der Bash realisieren kann. Ist die Bash bereit, dann erscheint meist ein Zeichen wie z.B. #. In dieser Zeile kann ein Benutzer dann seine Befehle eingeben - diese Zeile wird auch Kommandozeile genannt. Die Bash selbst ist ein Programm, welches ein Interface darstellt zwischen Mensch und Maschine. Deshalb ist diese auch mit einer Menge Features ausgestattet, welche es dem Benutzer einfach

machen, diverse Tätigkeiten auszuführen. Wird eine Shell gestartet (also z.B. die bash), dann werden grundlegende Einstellungen getätigt, so z.B. Umgebungsvariablen gesetzt. Erst dann ist die Shell bereit Eingaben anzunehmen.

7.1.2 Kommandozeileneditor

Gibt man also Befehle ein, so kann man einfach innerhalb der Kommandozeile navigieren, um etwa Tippfehler zu korrigieren . Dafür gibt es die folgenden Tastenkombinationen:

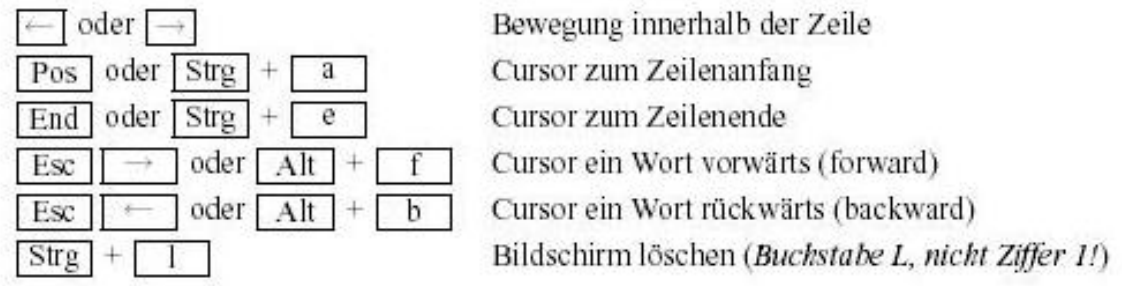
← oder →
Bewegung innerhalb der Zeile
Pos oder Strg + a
Cursor zum Zeilenanfang
End oder Strg + e
Cursor zum Zeilenende
Esc → oder Alt + f
Cursor ein Wort vorwärts (forward)
Esc ← oder Alt + b
Cursor ein Wort rückwärts (backward)
Strg + l
Bildschirm löschen (Buchstabe L, nicht Ziffer 1!)

Abbildung 7.1: Kommandozeileneditor Tasten [20]

7.1.3 Automatische Kommando- und Dateinamenerweiterung (Tabulator-Taste)

Wie bereits in Kapitel 5.1 angesprochen, kann man mittels der **Tabulator-Taste** Kommandos oder Dateinamen erweitern. Alle dem System bekannten Kommandos (genauer: diejenigen die sich im Pfad befinden) können automatisiert erweitert werden. Dies gilt ebenso für Datei- und Verzeichnisnamen.

Befindet sich z.B. im aktuellen Verzeichnis die Datei *brief_an_oma.txt* und man möchte sich diese per *less* anzeigen lassen, so genügt es einzugeben: `less b` und auf die Tabulatortaste zu drücken. Danach wird automatisch der Dateiname zu *brief_an_oma.txt* ergänzt. Gibt es allerdings noch eine weitere Datei *brief_finanzamt.txt* im selben Verzeichnis, so wird per Druck auf die Tabulator-Taste nur ergänzt auf `less brief_` - und nichts weiter. Durch einen schnellen zweimaligen Druck auf die Tabulator-Taste werden alle möglichen Dateien angezeigt, welche auf *less brief_* passen - eben die beiden Dateien.

7.1.4 History - das Gedächtnis der Kommandozeile

Jeder Befehl, den man eingegeben hat wird automatisch in der Datei *~/.bash_history* gespeichert (meist per Voreinstellung 500-1000 Befehle). Man kann einfach auf diese History zugreifen;

entweder über den eingebauten bash-Befehl *history*, welcher alle gespeicherten Befehle in einer nummerierten Liste darstellt. Will man auf Befehle zugreifen, welche man erst vor kurzem eingegeben hat, so bieten sich die folgenden Tastenkombinationen an:

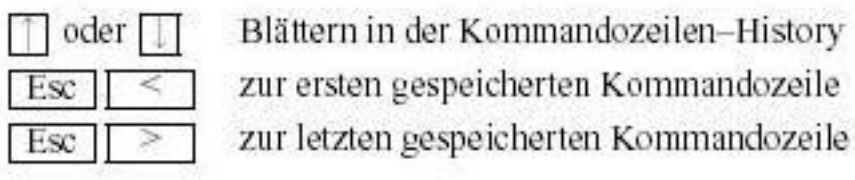

Abbildung 7.2: Bash History [20]

7.1.5 Reservierte Worte und Zeichen mit besonderer Bedeutung

In der Bash gibt es einige reservierte Worte und Zeichen, welche eine besondere Bedeutung haben, z.B. für die Shellprogrammierung. Diese Worte und Zeichen dürfen **nicht** in Dateinamen o.Ä. benutzt werden:

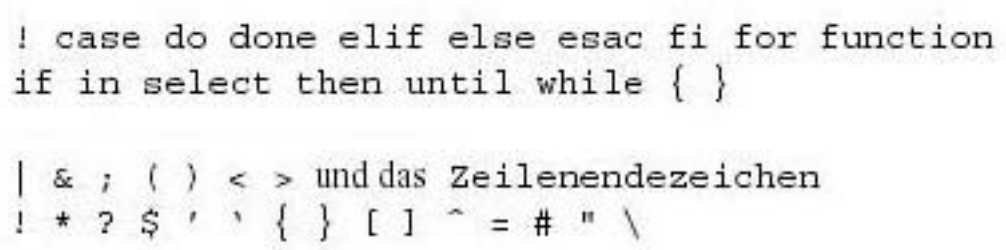

Abbildung 7.3: Bash Sonderzeichen [20]

7.1.6 Quotierung

Enthalten Datei- oder Verzeichnisnamen doch einmal Sonderzeichen, so kann man deren Bedeutung durch *Quotierung* ausschalten:

- **Fluchtsymbol \ (Backslash):** entwertet das unmittelbar folgende Sonderzeichen
- **Hochkomma ':** in Hochkommata eingeschlossener Text verliert seine Sonderbedeutung
- **Anführungszeichen ":** innerhalb von Anführungszeichen behalten nur das Dollarzeichen, Hochkomma und Backslash ihre Sonderbedeutung

7.1.7 Erweiterungen

Bei Erweiterungen muss man unterscheiden zwischen *Wildcards* und *Klammererweiterungen*.

Wildcards

Alle in der Kommandozeile eingegebenen Worte werden durchsucht nach den Zeichen * ? [ ]. Falls eines der Zeichen gefunden wird, wird das Wort als Muster für einen Pfad- oder Verzeichnisnamen genommen.

- * : passt auf alle Zeichenketten, einschließlich der leeren Zeichenkette
- ? : passt auf ein einzelnes Zeichen
- [ ] : passt auf alle in den eckigen Klammern angegebenen Zeichen; dabei können auch Bereiche angegeben werden wie z.B. [0-9] oder [a-z]

Beispiel: hat man im aktuellen Verzeichnis die Dateien *Buch Tuch Z. b.a bach berg buch*, dann wird folgendermaßen erweitert:

- `ls b?ch --> bach buch`
- `[A-Z]uch --> Buch Tuch`
- `ls b[ae]* --> bach berg`
- `ls *[^c]? --> Z. b.a berg`
- `echo 3 * 5 --> 3 Buch Tuch Z. b.a bach berg buch 5`

Es sind darüber hinaus auch Pfadnamen wie */ho*/an??/*te* möglich.

Klammererweiterung

Kommt in einem Wort eine geschweifte Klammer vor, welche eine durch Kommata getrennte Liste von Zeichenketten erhält, so wird das Wort durch deren Inhalt in mehreren Varianten ausgegeben. Beispiele:

- `echo alf{1,2,3} ! alf1 alf2 alf3`
- `echo M{a,i,o}st{1,2} ! Mast1 Mast2 Mist1 Mist2 Most1 Most2`

7.1.8 Wichtige Dateien für die Bash

Bevor die Bash gestartet wird, liest sie diverse Konfigurationsdateien aus, welche sich zum Teil im HOME-Verzeichnis des Benutzers befinden. In diesen Dateien werden grundlegende Einstellungen, Umgebungsvariablen, uvm. gesetzt. Die Dateien werden der Reihe nach gelesen, je nachdem ob sie existieren oder nicht. Dabei muss man auch noch unterscheiden zwischen einer *Login-Shell* und einer *Nicht-Login-Shell*.

Eine Login-Shell ist eine Shell, in der man sich initial am System einloggt, also z.B. durch Druck auf Strg-Alt-F1 bis -F6. Diese Shell wird als Login-Shell bezeichnet. Eine Nicht-Login-Shell ist eine Shell, welche man startet, wenn man bereits eingeloggt ist. Dies ist der Fall wenn man bereits KDE oder eine andere grafische Oberfläche gestartet hat und per Startmenü ein grafisches Terminal startet.

- **Login-Shell:** ~/etc/profile, ~/.bash_profile, ~/.bash_login, ~/.profile
- **Nicht-Login-Shell:** ~/.bashrc

Ändert man nun was an einer der Dateien, so muss man sich immer bewusst sein, dass man bei einer Login-Shell sich erneut einloggen muss damit die Änderungen gültig werden. Bei einer Nicht-Login-Shell genügt es ein neues Terminalfenster zu starten.

Nun stellt sich noch die Frage, in welcher Datei man was ändert. Will man z.B. Pfadvariablen (**PATH**) für alle Benutzter Systemweit ändern, so bietet sich die */etc/profile* an. Will man nur Benutzerpfade für einzelne Benutzer oder sich selbst ändern, so kann man das auch in den anderen genannten Dateien tun.

Etwas schwieriger wird es bei Aliasen: Aliase (siehe Kapitel 7.2) sollten unabhängig davon sein ob es sich um eine Login-Shell handelt oder nicht. Deswegen definiert man am besten Aliase doppelt, sowohl in der/den Datei(en) der Login-Shells als auch der Datei der Nicht-Login-Shell. Für genauere Infos hierüber empfiehlt sich [21], S.891.

7.2 Kommandos

7.2.1 Wichtige Shell Kommandos

In der Bash gibt es eine Anzahl bereits eingebauter Kommandos, d.h. diese Programme müssen nicht extern ausgeführt werden, sondern sind bereits innerhalb der Shell enthalten.

- **alias** legt ein Synonym fest, z.B. für Befehle

- **bg, fg** bringt ein Programm in den Hintergrund als Job, oder in den Vordergrund (siehe Kapitel 6.4)
- **echo** gibt eine Zeichenfolge zurück, z.B. `echo hallo`
- **exec** Programm ersetzt die aktuelle Shell durch das Programm, z.b. `exec bash`
- **exit** beendet die Shell
- **test -e** *datei* gibt „wahr“ zurück wenn die datei existiert
- **test -d** *datei* gibt „wahr“ zurück, wenn die datei ein Verzeichnis ist
- **test** *zahl1* **-le** *zahl2* gibt „wahr“ zurück, wenn zahl1 kleiner zahl2 ist

7.2.2 Exit-Status

Wenn man ein Programm in der Bash ausführt und dieses beendet wird oder sich selbst beendet, dann gibt es immer einen *Exit-Status* an die Bash zurück. Diesen Status kann man auswerten, oder sich den letzten Exit-Status anzeigen lassen mit `echo $?`. Genauer gesagt wird eine Integer-Zahl zurückgegeben:

- **Status 0 „null“:** wird dieser Status zurückgegeben, so wurde das Programm fehlerfrei beendet; anders als in der Programmiersprache C wird dies als „wahr“ interpretiert
- **anderer Status:** (z.B. 1,2,...) jede höhere Zahl bedeuted, dass ein Fehler aufgetreten ist und das Programm abnormal beendet wurde

7.2.3 Mehrere Kommandos in einer Kommandozeile

Man kann in einer Kommandozeile mehrere Kommandos nacheinander ausführen lassen, welche verschiedenen Regeln folgen:

- **befehl1 ; befehl2** führt befehl2 nach befehl1 aus
- **befehl1 && befehl2** führt befehl2 nur dann aus, nachdem befehl1 erfolgreich (Status=0) beendet wurde; Beispiel: `cp Vorlage MeineDatei && vi MeineDatei`
- **befehl1 || befehl2** führt befehl2 nur dann aus, nachdem befehl1 **nicht** erfolgreich beendet wurde; Beispiel: `rm datei` || `echo Fehler: Datei kann nicht gelöscht werden`
- **(befehl1 ; befehl2)** Die Kommandos werden in einer Subshell ausgeführt, Änderungen, z.B. des aktuellen Verzeichnisses, werden in der aktuellen Shell nicht wirksam

7.2.4 Arithmetische Ausdrücke Berechnen

Die Bash ist zwar ein mächtiges Werkzeug, doch ist sie nicht in der Lage, Berechnungen auszuführen. Wenn man z.B. 4+5 eingibt, so weiss die Bash nicht was Sie damit tun soll. Um eine Berechnung ausführen zu können muss man den Ausdruck in Eckige Klammern setzen, also z.B.

- `echo $[4+5]` - dies gibt das Ergebnis (9) aus.

Innerhalb der eckigen Klammern sind die meisten Operationen aus der Sprache C erlaubt, also `+ - * /` für die Grundrechenarten, `%` für Modulo, `== != < > <= >=` für Vergleiche, `<< >>` für Bitverschiebungen, `!`, `&&` und || für logisches NICHT, UND, ODER, etc.

Man kann diese Berechnungen auch einer Variable zuweisen, so z.B. `i=$[4+5]` und diese Variable dann per `echo $i` wieder ausgeben lassen.

7.2.5 Kommandosubstitition

Wenn man innerhalb einer Kommandozeile ein Ergebnis aus einem Kommando (also dessen *stdout*) benötigt, um dieses Ergebnis in ein anderes Kommando einzusetzen, so kann man dies mittels Kommandosubstitution tun. Aufmerksame Leser fragen sich jetzt bestimmt, wieso man dies nicht mittels einer | machen könnte - zu Recht. Oftmals ist es nicht möglich, ein *stdout* eines Kommandos direkt in den *stdin* des nächsten Kommandos umzuleiten. Oft wird eben das Ergebnis irgendwo „mittendrin“ im nächsten Kommando als Argument oder Option gebraucht.

Ein Beispiel zum besseren Verständnis: der folgende Ausdruck wäre mit Pipes nicht möglich; es soll zuerst im Verzeichnis */home/alex* nach allen Dateien gesucht werden in denen die Zeichenkette „studienarbeit*“ vorkommt. Dieses Ergebnis soll nun als Verzeichnisliste mittels `ls -l` ausgegeben werden. Die zwei folgenden Ausdrücke zum bewerkstelligen dieser Aufgabe sind absolut gleichwertig:

- `ls -l 'find /home/alex -name '*studienarbeit*''`
- `ls -l $(find /home/alex -name '*studienarbeit*')`

Das Ganze ist allerdings auch ohne Kommandosubstitution durch das Programm **xargs** möglich. Durch Zuhilfenahme von *xargs* werden Daten aus der Standardeingabe an das nach xargs angegebene Kommando weitergeleitet:

- `find /home/alex -name '*studienarbeit*' | xargs ls -l`

Würde man das Kommando *xargs* einfach weglassen, so funktioniert das Ganze nicht mehr wie gewünscht.

7.2.6 Aliase (Synonyme)

Mittels des Befehls *alias* kann man sich eine Menge Tipparbeit sparen: man kann Aliase für beliebige Befehle oder Befehlsfolgen vergeben und ist dabei so gut wie nicht eingeschränkt. Aliase kann man zur Laufzeit definieren oder in der Datei *∼/.bashrc* wo diese beim Start einer neuen Bash-Shell automatisch geladen werden. Wenn man einfach nur den Befehl *alias* auf der Bash angibt, so bekommt man alle bereits eingerichtete Aliase angezeigt. Will man selbst einen Alias definieren, so kann man das ganz einfach tun (hier einige Beispiele):

- `alias l='ls -lisa'` : dieser Alias würde bei Aufruf von „l“ den Befehl `ls -lisa` ausführen.
- `alias loginserv='ssh alex@linuxserver.muppets.heimnetz.de'` : wird man den Alias „*loginserv*“ aufrufen, so würde automatisch eine SSH-Verbindung zum entsprechenden Server aufgebaut

7.3 Variablen

Variablen spielen eine wichtige Rolle in einem Linux-System; seien es Variablen für Pfade, für Programme, Variablen zur Shell-Programmierung oder für andere Dinge.

7.3.1 Variablen - Setzen und Lesen

Eine Zuweisung an eine Variable erfolgt mittels **Name=Wert**. Auf beiden Seiten des „=“ Zeichens dürfen dabei keine Leerzeichen stehen. Um auf den Wert einer Variable zuzugreifen, bedient man sich **$Variable**. Man kann sich den Wert einer Variable auch wieder ausgeben lassen mittels echo, also z.B. `echo $Variable`.

Weiterhin sind auch *Felder* (Arrays) möglich: **Feld[2]=12345, echo $Feld[2]**.

Kindprozesse haben normalerweise keinen Zugriff auf Variablen des Vaterprozesses. Dies hat interne Systemgründe, welche zuletzt auch mit Sicherheit zu tun haben, aber an dieser Stelle nicht weiter erläutert werden. Diese Abgrenzung kann man jedoch umgehen, indem man die Variablen des Vaterprozesses *exportiert*, und zwar mittels **export Name=Wert**.

Man kann sich alle Umgebungsvariablen auch anzeigen lassen mittels des Befehls **set**.

7.3.2 Auswahl wichtiger Umgebungsvariablen

Es gibt eine unüberschaubare Vielzahl von Umgebungsvariablen; fast jedes Programm benötigt eigene Umgebungsvariablen, damit es funktioniert. Hier eine Auswahl sehr wichtiger Variablen welche in (fast) jedem Linuxsystem gesetzt sein sollten:

- **PATH :** der PATH, auch Suchpfad genannt, ist eine elementare Umgebungsvariable, in welcher sämtliche Wichtige Verzeichnisse stehen, die für den Gebrauch wichtige Programme enthalten sind.
- **HOME :** Pfad zum persönlichen Home-Verzeichnis des Benutzers
- **HISTSIZE :** Größe der maximalen Einträge in der Bash-History
- **PS1 :** Zeichenkette für die Eingabeaufforderung, z.B. `export PS1='\u@\h:\w >'` liefert: *Nutzer@Rechner: Verzeichnis*
- *** :** alle Positionsparameter
- **# :** Anzahl der Positionsparameter
- **? :** Exit-Status des zuletzt ausgeführten Kommandos
- **$:** Prozess-Nummer der aktuellen Shell

7.4 Shellprogrammierung

7.4.1 Wiederkehrende Tätigkeiten

So könnte man wohl die Grundidee für die Shellprogrammierung betiteln. Wiederkehrende Kommandos und Programme kann man in einer Textdatei zusammenfassen und dann in der Shell zur Ausführung bringen. Diese Textdateien nennt man dann *Shellskripte*. In diesen Skripten können Variablen benutzt werden, Schleifen und Verzweigungen programmiert und Funktionen definiert werden. Auf diese Weise ist man in der Lage, strukturiert programmieren zu können.

Ein Beispiel: um sich den aktuellen Benutzer und die aktuelle Uhrzeit und Datum anzeigen zu lassen, kann man eine Datei mit z.B. dem Namen *myinfo* anlegen, in welche man die drei folgenden Zeilen schreibt:

```
#! /bin/bash
who
date
```

Nun muss man noch mittels `chmod u+x myinfo` der Datei das Ausführungsrecht geben, dann kann man das einfache Skript starten mittels `./myinfo`. Die Shell erkennt, dass es sich um kein Binary handelt, sondern um ein Skript welches prozedurale Befehle enthält.

Die erste Zeile im Skript gibt an, welche Shell benutzt werden soll für die Ausführung der Befehle im Skript; in den meissten Skripts befindet sich hier an der Stelle auch der Eintrag für die Bash. Damit stellt man sicher, dass auf jedem System versucht wird, die Bash als Interpreter für das Skript zu benutzen. Man kann hier auch ein beliebiges anderes Programm zum interpretieren der Befehle im Skript angeben, so z.B. */usr/bin/perl* oder einen eigenen Interpreter. Das macht allerdings nur Sinn, wenn dieser Interpreter die Kommandos im Skript auch versteht. Die folgenden zwei Zeilen sind eben die Kommandos die ausgeführt werden sollen.

Im Prinzip können alle Programme des Systems und Programmierhilfen zur Gestaltung der Skipte behilflich sein, im folgenden sollen einige Möglichkeiten aufgeführt werden.

7.4.2 Kommentare

Kommentare können einfach mit dem Zeichen „#“ eingefügt werden, alles was zwischen # und Zeilenende steht wird als Kommentar aufgefasst.

7.4.3 Schleifen

Ähnlich wie in der Programmiersprache C, können Schleifen in der Bash verwendet werden. Wie in C, sind verschiedene Arten von Schleifen möglich, folgend in einigen Beispielen aufgegriffen. Schleifen können selbstverständlich nicht nur in Shell-Skripten verwendet werden, sondern auch direkt in der Kommandozeile, z.B. für kleinere Aufgaben. Die Shell erkennt selbständig, dass eine schleife noch nicht abgeschlossen ist und bietet einen *Fortsetzungprompt* an, d.h. wenn man die folgenden Beispiele einfach Zeilenweise eintippt, so kann man dies auch Zeilenweise auf der Shell tun. Alternativ kann man Schleifen auch in eine Kommandozeile eingeben mit folgender Schreibweise:

- `while [ ... ]; do BEFEHL; BEFEHL2; done`

Schleifen mit while

In diesem Beispiel werden in einer Schleife mehrere Verzeichnisse angelegt, nämlich */irgendwo/verz1* bis *verz9*. Die Zählvariable *i* wird pro Durchlauf in Zeile 8 inkrementiert. Die Schleife wird wiederholt, solange das hinter dem Schlüsselwort *while* stehende Kommando den Exit-Status

„null“ liefert. Anstelle der untenstehenden Bedingung könnte (wenn dies Sinn macht) auch jedes beliebige andere Kommando stehen: einzig der Exit-Status ist für die Wiederholung der Schleife maßgeblich.

```
1 #!/bin/bash
2 i=1
3 while [ $i -lt 10 ]
4 do
5   echo "Lege Verzeichnis /irgendwo/verz$i an ..."
6   mkdir /irgendwo/verz$i
7   i=$((i+1))
8 done
```

Schleifen mit until

Hier wird wie oben beschrieben die Schleifen-Bedingung durch den Exit-Status der Pipe beeinflusst: die Schleife läuft, bis *grep* erstmals das Wort *root* in der Liste der angemeldeten Benutzer findet.

```
1 #!/bin/bash
2 # Warten, bis sich der Admin einloggt:
3
4 until who | grep "root"
5 do
6   sleep 30       # 30s warten
7 done
8 echo "Big Brother is watching you!"
```

Aufzählungsschleife (for)

```
1 #!/bin/bash
2 for datei in *.c *.h
3 do
4   cp $datei $datei.bak
5 done
```

In Zeile 2 ist `datei` ein Variablenname: dieser Variable wird dann nacheinander jeder Wert aus der *Liste nach dem Schlüsselwort* `in` zugeordnet. Die Schleife wird damit einmal durchlaufen. Im

Beispiel befinden sich in der Liste **.c *.h* Wildcards; diese werden von der Shell im vorab durch passende Dateinamen (falls vorhanden) ersetzt. Man könnte durch Kommandosubstitution noch weiter gehen und die Liste durch eine Datei mit einer Liste ersetzen; Zeile 2 würde dann lauten: `for datei in $(cat meine_dateien.txt)` - somit werden die zu ladenden Dateien aus der Liste in *meine_dateien.txt* entnommen.

Zählschleife (for)

```
1 #!/bin/bash
2 for (( i=3 ; i<6 ; i++ ))
3 do
4   echo "$i $((i*i))"
5 done
```

Dies ist eine Variante der for-Schleife und eigenet sich besonders als Zählschleife. Diese Schleife gibt die natürlichen Zahlen 3 bis 5 mit ihren Quadraten aus.

7.4.4 Verzweigungen

Bedingte Verzweigungen

```
1 #!/bin/bash
2 ...
3 if grep "abc" datei
4 then
5   echo "Muster gefunden"
6 else
7   echo "nix da"
8 fi
```

Für die Bedingung, die in Zeile 3 hinter if steht gilt alles was dazu im Abschnitt „Schleifen mit while“ gesagt wurde: auch hier entscheidet der Exit-Status, welcher Zweig durchlaufen wird.

Mehrfache Verzweigungen

```
1 #!/bin/bash
```

```
2 ...
3 system=linux     # sollte beim Ablauf des Scriptes geeignet gesetzt werden
4 ...
5       # jetzt wird $system ausgewertet
6 case $system in
7 linux)     echo "wir schaffen unter Linux" ;;
8 aix|hpux)   echo "das klingt nach IBM oder HP" ;;
9 *)         echo "noch was anderes" ;;
10 esac
```

Hier wird der Wert der Variablen *system* nacheinander mit den angegebenen Mustern verglichen. Der/ Die Befehl(e) beim ersten Treffer werden ausgeführt und spätere Treffer nicht mehr berücksichtigt. Die Befehlsabarbeitung wird hinter *esac* fortgesetzt.

Die Muster werden jeweils mit einer) abgeschlossen und können Wildcards enthalten (in Zeile 9: der Stern passt immer). Außerdem kann man in einer Zeile mehrere durch | getrennte Muster angeben (im Beispiel *aix* oder *hpux*). Das Ende einer Befehlssequenz, die vollständig zu einem Muster gehört, wird durch ;; markiert.

7.4.5 Funktionen

```
1 ...
2 Fehler()
3 {
4   exitstatus=$1
5   message=$2
6   echo "FEHLER: $message" >2
7   exit $exitstatus
8 }
9 ...
```

Danach kann man die Funktion aufrufen mittels:

```
[ -e "hello.c" ] || Fehler 2 "Datei nicht gefunden"
```

Falls im Beispiel die Datei *hello.c* nicht existiert, so erfolgt ein Aufruf der Funktion *Fehler* mit zwei Argumenten. Die Funktion gibt das zweite Argument (Zugriff ähnlich wie auf Argumente in der Kommandozeile) auf stdout aus und beendet das Programm anschließend mit Exit-Status „2“.

8 Der Editor vi

Für alle UNIX- und Linux-Systeme ist ein Text-Editor essentiell, da die Grundlage für die Konfiguration dieser Systeme fast ausschließlich textbasierte Konfigurationsdateien sind. In diesem Kapitel soll Ihnen die Funktionweise und die wichtigsten Kommandos zum Editor vi, der „klassische“ Text-Editor bei UNIX- und Linux-Systemen, näher gebracht werden. Aufgrund seiner ungewohnten Bedienung schrecken viele Anfänger im Umgang mit Linux zurück vor der Verwendung dieses Editors. Hat man sich aber ein wenig an die Funktionsweise gewöhnt, erkennt man die Effizenz und Leistungsstärke dieses Editors.

Die folgenden Punkte sind der Inhalt dieses Kapitels.

- 8.1 Allgemeines
- 8.2 Modi & Kommandos

8.1 Allgemeines zum Editor vi

Der Editor vi ist der Standard-Editor für alle auf UNIX basierenden Systeme. Daher wird auch bei jeder Linux-Distribution der Editor vi mitgeliefert oder ist für diese verfügbar und kann bei Bedarf nachinstalliert werden (wenn nicht schon Teil der Standard-Installation). Aus lizenzrechtlichen Gründen handelt sich bei Linux jedoch nicht um den „originalen“ vi, sondern um vi-kompatible Editoren wie zum Beispiel die Editoren vim und elvis. Die größte Verbreitung im Linux-Umfeld genießt sicherlich der Editor vim. Dies ist jedoch für Sie als Benutzer transparent, da auch die vi-kompatiblen Editoren in gleicher Weise auf der Kommandozeile aufgerufen werden wie der ursprüngliche Editor vi:

```
vi
```

Daher wird „vi“ umgangssprachlich meist als Synonym für den ursprünglichen Editor vi, als auch alle seine „Ableger“ verwendet (so auch im folgenden Text). Anschließend erhalten Sie in der Kommandozeile eine Art „Startbildschirm“ wie er in folgender Abbildung zu sehen ist.

Der Editor vi hat kein grafisches Bedienungsmenü (wie Sie es vielleicht von anderen Editoren

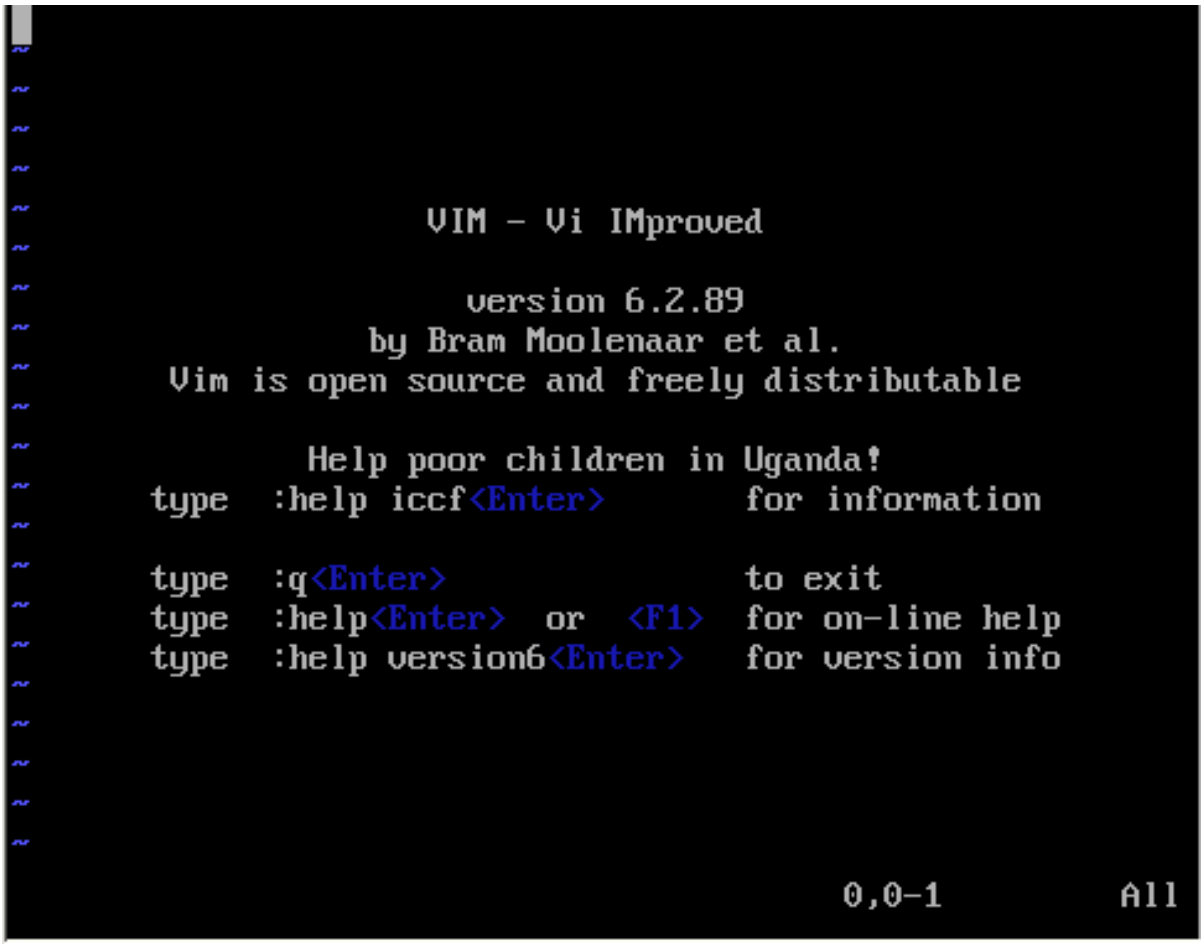

Abbildung 8.1: Startbildschirm des Editor vi

kennen) und wird lediglich über Tastatureingaben gesteuert. Man könnte sich an dieser Stelle die Frage stellen, warum in diesem Kurs gerade dieser unter so vielen für Linux verfügbaren Editoren vorgestellt wird. Dazu sei an dieser Stelle folgendes angemerkt:

- Der Editor *vi* ist vergleichsweise kompakt und steht eventuell auch dann zur Verfügung, wenn kein anderer Editor im Moment nutzbar ist (etwa bei Wartungsarbeiten, wenn Linux von einer Notfalldiskette gebootet wurde.

- Wie zuvor schon angedeutet, steht der Editor vi unter allen UNIX- und Linux-Systemen zur Verfügung und stellt insofern einen Standard dar (also unabhängig davon, ob Sie mit Red Hat Linux, SuSE Linux oder IBM AIX arbeiten). Aus diesem Grund wird der Editor vi von diversen Programmen automatisch als Editor aufgerufen, wenn dies nicht explizit durch eine entsprechende Konfiguration verhindert wird. In diesen Fällen sollten Sie wenigstens wissen, wie Sie den Editor verlassen können.

In der Mitte der Ansicht finden Sie eine Auswahl grundlegender Kommandos und deren Bedeutung. Rechts am unteren Rand werden beim Editor vi die Zeilen- und die Zeichenposition des Cursors im Format „*Zeile,Zeichenposition*“ angezeigt. Am linken Rand sind alle Zeilen unter der Zeile in der sich der Cursor befindet mit einer Tilde („∼“) markiert. Auf diese Weise werden leere Zeilen, die weder normale Zeichen noch Leerzeichen enthalten, markiert. In den meisten Fällen werden Sie sicherlich den Editor vi starten, um eine oder mehrere Textdateien zu öffnen. Dies können Sie durch Anführen der Dateinamen nach dem vi-Kommando:

`vi` *textdatei1.txt textdatei2.txt*

Sie haben aber auch die Möglichkeit einen beliebigen Dateinamen anzugeben (zu dem also keine Datei existiert). In diesem Fall wird eine neue temporäre Textdatei mit entsprechendem Namen erzeugt, die aber erst nach dem Speichern im Dateisystem abgelegt wird. Dies erkennen Sie im Editor vi durch die Anzeige „[New File]“ hinter dem Dateinamen (hier: *textdatei.txt*) am unteren Rand.

Abbildung 8.2: Anzeige „New File“ im Editor vi

Bei einer vorhandenen Datei wird stattdessen hinter dem Dateinamen die „Größe“ (hier: 1L, 24C) angezeigt. Dabei handelt sich um die Anzahl der Zeilen (engl. **L**ines) und der Zeichen (engl. **C**haracters), die die Datei enthält (Abbildung 8.3).

8.2 Modi & Kommandos

Nachdem Sie im letzten Abschnitt einen Überblick über die Herkunft, das Starten und die Ansichtsinformationen des Editor vi bekommen haben, soll in diesem Abschnitt nun das Arbeiten mit diesem Editor Ihnen näher gebracht werden.

8.2.1 Die drei Modi des Editor vi

Der wichtigste Unterschied zu anderen Editoren ist, dass der Editor vi verschiedenen Modi unterscheidet, die alle durch Tastatureingaben gewählt werden können:

Eingabemodus: Für die Eingabe von neuem Text ist dieser Modus vorgesehen. Jede Tastatureingabe wird Zeicheneingabe von Editor interpretiert und in den Text eingefügt.

Abbildung 8.3: Anzeige der Zeilen und Zeichen im Editor vi

Kommandomodus: Der Kommandomodus ist der Ausgangspunkt beim Starten des Editors beziehungsweise beim Öffnen einer Textdatei mit dem Editor. Mit der Escape-Taste kehrt man aus dem Eingabemodus in diesen Modus zurück, aus dem Zeilenmodus nach erfolgreicher Eingabe eines Befehls. Der Kommandomodus ermöglicht das schnelle Durchführen größerer Änderugen im zu editierenden Text (zum Beispiel das Löschen von 5 Zeilen ab der Cursor-Position: „`5dd`“).

Zeilenmodus: Operationen auf Datei-Ebene (zum Beispiel das Speichern der Änderungen an der gerade geöffneten Datei: „`:w`“ sind in diesem Modus möglich.

Abbildung 8.4 gibt Ihnen nochmal einen Überblick über die vorgestellten Modi in Form eines Graphen.

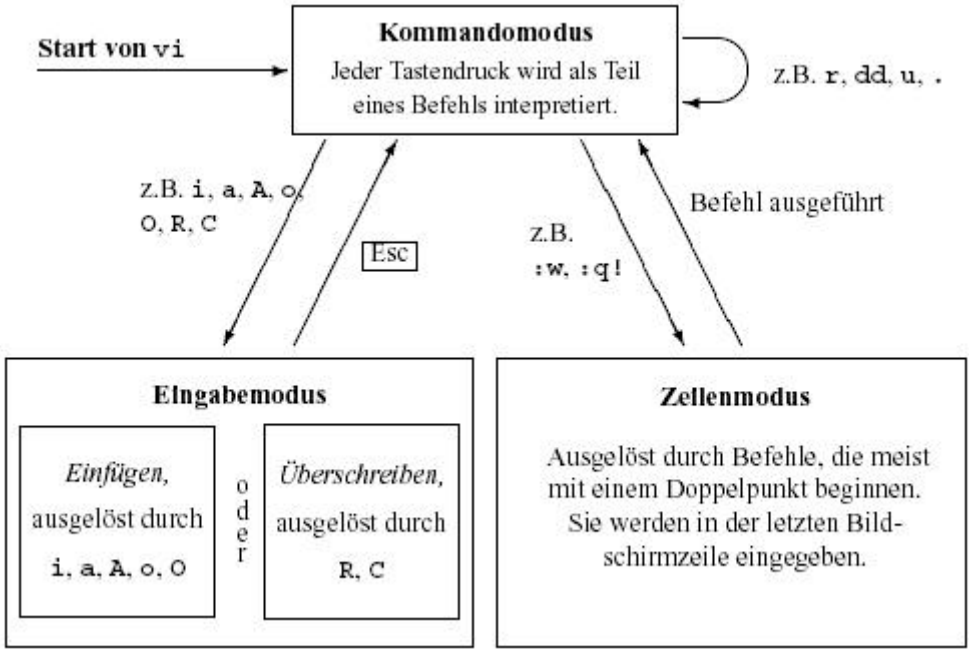
Start von vi
Kommandomodus
Jeder Tastendruck wird als Teil eines Befehls interpretiert.
z.B. r, dd, u, .
z.B. i, a, A, o, O, R, C
Esc
z.B. :w, :q!
Befehl ausgeführt
Eingabemodus
Einfügen, ausgelöst durch i, a, A, o, O
oder
Überschreiben, ausgelöst durch R, C
Zeilenmodus
Ausgelöst durch Befehle, die meist mit einem Doppelpunkt beginnen. Sie werden in der letzten Bildschirmzeile eingegeben.

Abbildung 8.4: Die Modi des Editor vi [20]

8.2.2 Wichtige Befehle im Kommandomodus

Die folgende Übersicht (Abbildung 8.5) stellt eine Auswahl der wichtigsten Befehle im Kommandomodus dar.

8.2.3 Wichtige Befehle im Zeilenmodus

Die folgende Übersicht (Abbildung 8.6) stellt eine Auswahl der wichtigsten Befehle im Zeilenmodus dar.

Wichtig für das „flüssige“ Arbeiten mit dem Editor vi ist der gewohnte Umgang mit den verschiedenen Modi und deren Kommandos. Probieren Sie ruhig das ein oder andere Kommando aus, um ein wenig Routine im Umgang mit diesem Editor zu bekommen. Weitere Dokumentation zum Editor vi finden Sie unter `http://www.vim.org/docs.php`.

I	⟶ Eingabemodus: Zeichen werden *an* aktueller Cursorpostion eingegeben
a	⟶ Eingabemodus: Zeichen werden *nach* aktueller Position eingegeben
A	⟶ Eingabemodus: Zeichen werden am Ende der Zeile angehängt
C	⟶ Eingabemodus: der Rest der Zeile wird durch den neuen Text ersetzt
o	⟶ Eingabemodus: *nach* der aktuellen Zeile wird neue Zeile eingefügt
O	⟶ Eingabemodus: *vor* der aktuellen Zeile wird neue Zeile eingefügt
R	⟶ Eingabemodus: überschreibt den alten Text
r	überschreibt ein einzelnes Zeichen
x	löscht das aktuelle Zeichen
dd	löscht die aktuelle Zeile (und stellt sie in den Puffer)
dw	löscht bis zum Ende des aktuellen Wortes (und stellt es in den Puffer)
u	nimmt den letzten Befehl zurück (undo)
J	hängt die folgende Zeile an die aktuelle an
yy	kopiert eine Zeile in einen Puffer
P bzw. **p**	fügt Puffer vor bzw. nach dem Cursor ein
1G	springt an den Dateianfang
G	springt ans Dateiende
^G	zeigt die Position innerhalb der Datei an (Strg + G)
%	steht der Cursor auf einer Klammer, so springt er daraufhin zur korrespondierenden Klammer
.	wiederholt den letzten Befehl
:	wechselt in den Zeilenmodus

Abbildung 8.5: Befehle im Kommandomodus [20]

:w [*Dateiname*]	speichert [unter *Dateiname*]
:q	verläßt den Editor (nach dem Speichern)
:q!	verläßt Editor ohne zu speichern
:x	entspricht `:wq`
:e *Dateiname*	editiert die entsprechende Datei
:u	nimmt den letzten Editierbefehl zurück
/Zeichenkette	sucht vorwärts nach *Zeichenkette*
?*Zeichenkette*	sucht rückwärts nach *Zeichenkette*
:r *Dateiname*	die Datei nach dem Cursor einfügen

Abbildung 8.6: Befehle im Zeilenmodus [20]

9 Externe Kommandos

Während *interne Kommandos* direkt von der Shell interpretiert werden, handelt es sich bei *externen Kommandos* genau genommen um eigenständige Programme die in der Shell ausgeführt werden.

Externe Kommandos werden im Suchpfad erwartet. So ist *cd* ein internes Kommando und *ls* ein externes Kommando. Das Programm zum Kommando *ls* findet sich im Regelfall im Verzeichnis */bin*.

In diesem Kapitel soll auf einige externe Kommandos eingegangen werden welche oft im Umgang mit Linux gebraucht werden.

Die folgenden Punkte sind der Inhalt dieses Kapitels; die hier genannten Befehle stellen nur eine Übersicht über die wichtigsten Befehle der nachfolgenden Kapitel dar.

- 9.1 Compilieren eines Quelltextes (gcc)
- 9.2 grep
- 9.3 Reguläre Ausrücke
- 9.4 gzip und gunzip
- 9.5 tar
- 9.6 mtools
- 9.7 find
- 9.8 Weitere Kommandos
- 9.9 Weitere Beispiele

9.1 Compilieren eines Quelltextes

9.1.1 Das Kommando gcc

Im Laufe Ihres Studiums werden Sie öfters in die Verlegenheit kommen C-Sourcen zu compilieren. An dieser Stelle sollen die grundlegenden Funktionalitäten des externen Kommandos (Programmes) *gcc* erläutert werden, welches Ihnen eben das ermöglicht.

GCC ist eigentlich eine Abkürzung für eine Sammlung mit der Bedeutung *GNU C Compiler*. Da GCC aber heute mehr als nur einen C-Compiler beinhaltet spricht man mittlerweile von *GNU Compiler Collection*. Die Bezeichnung *gcc* (klein geschrieben) steht weiterhin für den C-Compiler. Dieser Kurs wird sich ausschließlich mit dem C-Compiler beschäftigen.

Die meisten Linux-Distributionen installieren *gcc* automatisch mit. Um festzustellen ob *gcc* bei Ihnen korrekt installiert ist geben Sie folgendes Kommando in eine Shell ein:

```
gcc -v
```

```
rammstein:/etc # gcc -v
Reading specs from /usr/lib/gcc-lib/i486-suse-linux/3.3/specs
Configured with: ../configure --enable-threads=posix --prefix=/usr --wi
th-local-prefix=/usr/local --infodir=/usr/share/info --mandir=/usr/shar
e/man --libdir=/usr/lib --enable-languages=c,c++,f77,objc,java,ada --di
sable-checking --enable-libgcj --with-gxx-include-dir=/usr/include/g++
--with-slibdir=/lib --with-system-zlib --enable-shared --enable-__cxa_a
texit i486-suse-linux
Thread model: posix
gcc version 3.3 20030226 (prerelease) (SuSE Linux)
rammstein:/etc #
```

Abbildung 9.1: Konsolenausgabe des Kommandos gcc -v

gcc gibt aus wie es konfiguriert wurde und um welche Version es sich handelt. So werden u.a. genannt, mit welchem Threading-Modell und welchen Optionen gcc erstellt wurde (configured with, Thread model).

9.1.2 Sourcedatei kompilieren

Um eine vorhandene C-Source Datei zu kompilieren müssen Sie folgendes Kommando ausführen:

```
gcc meinesource.c -o meinlaufendesprogramm
```

Die Sourcedatei *meinesource.c* wird nun kompiliert und gelinkt. Am Ende steht eine ausführbare

Datei mit dem Namen *meinlaufendesprogramm*, so keine Programmierfehler vorliegen.

Sie können das Programm jetzt mit dem Befehl

```
./meinlaufendesprogramm
```

starten.

Weitere Parameter

gcc kann mit einer schon fast unübersichtlichen Anzahl von Parametern konfiguriert werden. An dieser Stelle folgen einige häufig gebrauchte. Weitere Informationen können Sie sich hier wie gewohnt über die Man Pages (Kapitel 3.1) abrufen.

- **-c:** Quelltexte übersetzen aber nicht linken
- **-I:** Pfad für eingebundene Header-Dateien
- **-L:** Pfad zu verwendeten Programmbibliotheken
- **-O2:** Compiler soll den Code weiter optimieren
- **-g:** Symboltabelle zum Debuggen an den erzeugten Code anfügen
- **-Wall:** Ausgabe detailierterer Warnungen
- **-std=:** Angabe des zu verwendenden Sprachstandards (mit *-std=c99* wird z.B. der iso9899:1999 verwendet)
- **-traditional:** verwende ANSI C Standard

Weitere Beispiele

```
gcc -Wall -g3 meinesource.c -o meinlaufendesprogramm
```

Die Sourcedatei *meinesource.c* wird wie oben kompiliert und gelinkt. Zudem werden durch die Verwendung des Paramters *-Wall* detaillierte Warnungen zusätzlich für folgende Fälle ausgegeben

- es werden Arrays mit Indizes vom Typ char definiert oder verwendet, entspricht dem Paramter *Wchar-subscrips*
- es wurde der Beginn eines verschachtelten Kommentars gefunden, entspricht dem Paramter *Wcomment*

- inkorrekt formatierte printf und scanf Aufrufe, entspricht dem Paramter *Wformat*
- Funktionen oder Paramter wurden implizit definiert, entspricht dem Paramter *Wimplicit*
- Funktionen sind ohne oder mit falschen Rückgabewert definiert, entspricht dem Paramter *Wreturn-type*
- Warnung bei switch-Anweisungen die Aufzählungstypen verwenden, bei denen nicht alle Werte des Aufzählungstyps auftreten, entspricht dem Paramter *Wswitch*
- Warnug wenn Debugging-Informationen für C++ Templates nicht zur Verfügung stehen, entspricht dem Paramter *Wtemplate-debugging*
- Variabeln wurden nicht initialisiert, entspricht dem Paramter *Wunitialized*
- Variabel oder Funktion wird nicht benutzt, entspricht dem Paramter *Wunused*

Außerdem werden durch den Paramter *g3* Debugging-Informationen mit eingefügt mit deren Hilfe das Programm später Schritt für Schritt durchlaufen werden kann. Der Wert 3 gibt dabei den Detailgrad der Debugging-Informationen an (Werte zwischen 1-3 sind zulässig).

```
gcc -O2 -std=c99 meinesource.c -o meinlaufendesprogramm
```

Mit dem Aufruf dieses Kommandos wird gcc angewiesen den iso9899:1999 Standard zu verwenden. Dies ist vor allem dann notwendig, wenn sie Erweiterungen dieses Standards verwenden. Außerdem wird der Quelltext durch den Parameter *O2* optimiert. Der Wert 2 gibt dabei an wie stark die Optimierung ist. (Werte zwischen 1-3 sind zulässig).

9.2 grep

Mit dem Kommando *grep* können Sie einen Textstream nach dem Auftreten von bestimmten Ausdrücken durchsuchen. *grep* gibt alle Zeilen in denen die gesuchten Ausdrücke auftreten aus. Sie können mit *grep* eine oder mehrere Dateien durchsuchen. Die Standardeingabe von *grep* kann auch auch in Pipes verwendet werden, z.B. `ls -lisa | grep "Stuttgart"`. Dieses Beispiel gibt nur die Verzeichnisinhalte aus, welche „Stuttgart“ beinhalten.

```
grep [Optionen] Ausdruck [Datei...]
```

Möglich sind u.a. folgende Optionen

- **-i** ignoriert Groß- und Kleinschreibung
- **-n** gibt die Zeilennummer zu jeder Fundstelle aus

- **-w** gibt nur Zeilen aus, die den Ausdruck als ganzes Wort enthalten
- **-C** gibt auch zwei Zeilen vor und nach der Fundstelle aus
- **-c** gibt Anzahl der Trefferzeilen aus
- **-q** unterdrückt die normale Ausgabe und bricht an der ersten Fundstelle ab
- **-s** unterdrückt Fehlermeldungen über nicht vorhandene ode nicht lesbare Dateien
- **-v** gibt alle Zeilen aus auf die das Suchmuster nicht passt
- **-x** gibt Zeilen nur aus wenn auf die gesamte Zeile das Suchmuster passt
- **-l** gibt Dateien mit Treffern aus, aber keinen detailierten Informationen, die Suche wird in einer Datei nach dem ersten Treffer abgebrochen

Beispiele:

```
grep -w Stuttgart MeinText.txt
```

Gibt alle Zeilen in der Datei *MeinText.txt* aus, die den Begriff *Stuttgart* als ganzes Wort enthalten

```
grep -i ga MeinText.txt AndererText.txt
```

Gibt alle Zeilen in den Dateien *MeinText.txt* und *AndererText.txt* aus die den Begriff *ga*, *gA*, *Ga* oder *GA* enthalten. So wird auch das Wort *Stuttgart* gefunden.

```
who | grep root
```

In diesem Fall wird nicht eine Datei durchsucht sondern die Standardeingabe von grep. Das Kommanso *who* liefert alle am System angemeldeten Benutzer zurück. Mittels Pipe wird die Ausgabe auf *grep* umgeleitet. Ist der Nutzer root angemeldet so gibt *grep* die entsprechende Zeile aus.

```
grep -l '^{}#include' /usr/include/*
```

Dieses Kommando gibt alle Dateien im Verzeichnis */usr/include* aus die mindestens einmal die Direktive *#include* enthalten.

9.3 Reguläre Ausdrücke

Eine Vielzahl von Programmen, wie auch *grep*, können *reguläre Ausdrücke* verarbeiten. Im folgenden werden einige Vertreter der *regulären Ausdrücke* kurz vorgestellt.

- **. :** Der Punkt symbolisiert ein beliebiges Zeichen.
- **+ :** Der voranstehende Ausdruck ist optional, er kann einmal vorkommen, muss es aber nicht, d.h. der Ausdruck kommt null- oder einmal vor.
- *** :** Der voranstehende Ausdruck muss mindestens einmal vorkommen, darf aber auch mehrfach vorkommen.
- **? :** der voranstehende Ausdruck darf beliebig oft (auch keinmal) vorkommen.
- **{n} :** Der voranstehende Ausdruck muss exakt *n*-mal vorkommen.
- **{min,max} :** Der voranstehende Ausdruck muss mindestens *min*-mal und darf maximal *max*-mal vorkommen.
- **{min,} :** Der voranstehende Ausdruck muss mindestens *min*-mal vorkommen.
- **^ :** Der Ausdruck passt auf einen Zeilenanfang (steht am Zeilenanfang)
- **$:** Der Ausdruck passt auf eine Zeilenende (steht am Zeilenende)
- **[Bereich] :** Mit eckigen Klammern lässt sich eine Zeichenauswahl definieren. Der Ausdruck in eckigen Klammern steht dann für genau ein Zeichen aus dieser Auswahl (Einzeichenmuster). ([a-m], [egh], [^a], [135] oder [A-Za-z0-9])
- **| :** Alternative („j|Ja“ passt auf „j“ und „Ja“)
- **d :** Passt auf Ziffern
- **D :** Passt nicht auf Ziffern
- **w :** Alle alphanumerischen Zeichen (Buchstaben, Ziffern und „_“)
- **W :** Alle **nicht** alphanumerischen Zeichen
- **s :** Whitespace-Zeichen, meistens [\f\n\r\t\v]
- **S :** Kein Whitespace-Zeichen
- **m :** Passt auf Wort-Anfang
- **M :** Passt auf Wort-Ende
- \ **:** Entwertet Sonderzeichen (*)

```
index.html
idiot.html
idiot.htm

haus
hans
hansel
hannes
mausi
hase
haas

x
xx
xxxx
xxxxxx
xyxyxy

.alfa
.mm
.num734
```

Abbildung 9.2: Beispieldatei *lugmu.txt*

9.3.1 Beispiele

Die folgenden Beispiele beziehen sich auf die Datei *lugmu.txt* (siehe Abbildung 9.2).

```
[0-9]+\. [A-Z][a-z]* 19[0-9][0-9]$
```

Passt auf ein Datum zwischen 1900 und 1999 welches am Zeilenende steht.

```
grep x lugmu.txt
```

findet alle Zeilen in *lugmu.txt*, die ein „x“ enthalten.

```
grep 'ha.s' lugmu.txt
```

findet alle Zeilen welche die Zeichenfolge „ha.s“ enthalten, wobei „.“ genau ein Zeichen ersetzen kann.
→ haus, hans, hansel, haas

```
grep '^m.*i$' lugmu.txt
```

findet alle Zeilen welche mit „m“ beginnen und mit „i“ aufhören.
→ mausi

```
grep 'h[aeiou]s' lugmu.txt
```

findet eine Zeichenfolge, bestehend aus 3 Zeichen, welche mit “h“ beginnt, irgend einen Vokal besitzt und mit „s“ aufhört.
→ hase

```
grep '^\.' lugmu.txt
```

findet Zeilen welche mit einem Punkt beginnen.

```
grep 'x{4}' lugmu.txt
```

findet eine Zeichenfolge mit vier „x“.

9.4 gzip und gunzip

Mit den beiden Kommandos *gzip* und *gunzip* können Sie Dateien komprimieren und wieder auspacken.

9.4.1 gzip

gzip komprimiert die angegebenen Dateien mit der Lempel-Ziv-Codierung (LZ77) dabei werden Zugriffsrechte und Zeitstempel der Dateien beibehalten. Symbolische Links werden durch gzip ignoriert.

```
gzip -9 index.html start.html
```

Mit diesem Befehl komprimieren sie die Datei *index.html* und *start.html*. Als Ergebnis erhalten Sie die Dateien *index.html.gz* und *start.html.gz*. Die Dateien werden also umbenannt indem die Endung *gz* angehangen wird. Der Parameter -9 gibt dabei den Grad der Komprimierung an, hier sind Werte zwischen 9 und 1 erlaubt wobei 9 für die beste Komprimierung und 1 für die schlechteste, aber auch schnellste Komprimierung steht.

```
gzip -9 -r sources
```

Mit diesem Befehl können sie alle Dateien des Verzeichnisses *sources* rekursiv komprimieren.

```
gzip -t index.html.gz
```

Dieser Befehl prüft die gepackte Datei *index.html.gz* auf Fehler.

Weitere Parameter

- **-q** unterdrückt alle Meldungen
- **-f** erzwingt die Durchführung der Komprimierung (keine Rückfrage wenn z.B. symbolische Links vorliegen)
- **-r** komprimiert rekursiv den Inhalt eines Verzeichnisses
- **-t** testet komprimierte Dateien auf Fehler
- **-v** gibt bei jeder Datei die erreichte Kompressionsrate und den Namen aus
- **-c** schreibt die komprimierten Daten auf die Standardausgabe und ersetzt nicht die Quelldatei.

9.4.2 gunzip

gunzip ist das Gegenstück zu gzip und entpackt Dateien wieder. Es verwendet die selben Paramter wie *gzip*.

```
gunzip index.html.gz
```

Durch dieses Kommando entpacken Sie die Datei *index.html.gz* wieder. Als Ergebniss erhalten Sie die Datei *index.html*.

9.4.3 Alternativen

Alternativ zu den Kommandos *gzip* und *gunzip* können Sie auch die Kommandos *bzip2* und *bunzip2* verwenden. Diese funktionieren ähnlich verwenden aber einen effektiveren Algorithmus und erreichen somit eine höhere Komprimierung. Als Dateiendung wird hier *bz2* verwendet.

Neben *bzip2* und *bunzip2* können Sie auch noch die Kommandos *compress* und *uncompress* verwenden. Hier wird die Dateiendung *Z* verwand.

9.5 tar

Mit dem Befehl *tar* können Dateiarchive verwaltet werden. In ein Archiv können Dateien, auf Wunsch auch komprimiert, in eine einzige große Datei verpackt werden. Sehr oft werden Programmpakete der Linux-Welt als Datei-Archive ausgeliefert.

Dateiarchive ohne Komprimierung tragen die Endung *.tar*.
Dateiarchive mit Komprimierung tragen die Endung *.tgz* oder *.tar.gz*.

9.5.1 Beispiele

```
tar czvf Backup.tgz /home/student
```

verpackt alle Dateien des Verzeichnisses */home/student* in ein komprimiertes Archiv *Backup.tgz* (Unterverzeichnisse werden mit verpackt).

```
tar tvf Backup.tgz
```

gibt alle Dateien aus die im Archiv *Backup.tgz* enthalten sind

```
tar xzvf Backup.tgz
```

packt das gesamte Archiv aus.

```
tar xvf Backup.tgz Brief
```

packt nur die Datei *Brief* aus dem Archiv *Backup.tgz* aus.

9.5.2 Funktionsoptionen

Genau eine der folgenden Optionen muss vor allen anderen Optionen angegeben werden.

- **-c** legt neues Archiv an
- **-d** vergleicht im Archiv abgelegt Dateien mit im Befehl angegebenen Dateien. Alle Unterschiede wie fehlende Dateien, unterschiedliche Größen, Zugriffsrechte u.s.w werden ausgegeben
- **-r** Dateien an das Ende des Archives anhängen
- **-t** Ausgabe der Dateien die im Archiv abgelegt sind

- **-u** Dateien zum Archiv hinzufügen wenn Sie sich noch nicht darin befinden oder sich geändert haben
- **-x** Dateien aus dem Archiv extrahieren

9.5.3 Weitere nützliche Optionen

- **-f** *Archiv* verwendet den mit Archiv angegebenen Dateinamen für das Archiv
- **-k** beim extrahieren werden Dateien mit ähnlichen Dateinamen nicht überschrieben, es wird eine Fehlermeldung ausgegeben
- **-m** die Zeit der letzten Modifikation wird nicht restauriert sondern auf den Zeitpunkt des Entpackens gesetzt
- **-p** Besitzer und Gruppen bleiben wie im Original
- **-v** Dateinamen ausgeben die einem Archiv hinzugefügt oder von einem Archiv extrahiert werden
- **-w** jede Aktion vom Anwender bestätigen lassen
- **-z** Dateien vor der Archivierung mit gzip komprimieren lassen oder beim Extrahieren mit gunzip dekomprimieren lassen
- **-N** *Datum* Dateien ignorieren die älter als das angegebene Datum sind
- **-W** Archiv nach Beendigung der Archivierung auf Integrität prüfen
- **-P** keine Schrägstriche am Anfang von Dateinamen entfernen
- **-Z** Dateien vor der Archivierung mit compress komprimieren oder vor dem extrahieren mit *uncompress* dekomprimieren

9.6 mtools

Bei den *mtools* handelt es sich um ein Programmpaket mit dessen Hilfe man auf DOS-Disketten zugreifen kann. Dabei entspricht die Namensgebung der Befehle denen unter DOS, nur das jeweils ein *m* vorangestellt wird.

Ein kleine Auswahl der wichtigsten Befehle:

- **mdir** gibt den Inhalt des aktuellen Verzeichnisses aus

- **mcopy [-t] Quelle Ziel** Kopiert Dateien vom angegebenen Quellverzeichnis in das angegeben Zielverzeichnis. Wird die Option *-t* gesetzt so werden Textformate zwischen Unix und DOS konvertiert
- **mdel** löscht eine Datei auf der Diskette
- **mcd** wechselt das Verzeichnis auf der Diskette

9.7 find

Das Kommando find sucht unter verschiedenen Kriterien Dateien und kann mit den gefundenen Dateien wenn gewünscht auch Aktionen ausführen.

```
find [Verzeichnis] [Test] [Aktion]
```

Der Verzeichnisbaum wird ab dem angegebenen Parameter rekursiv durchsucht. Gefunden werden die Dateien für die der Paramter Test erfüllt ist.

9.7.1 Nützliche Paramter

- **-name** *Muster* findet eine Datei deren Name dem *Muster* entspricht
- **-iname** *Muster* findet eine Datei deren Name dem *Muster* entspricht ohne Unterscheidung von Groß- und Kleinschreibung
- **-user** *Name* findet Dateien deren Eigentümer der Benutzer *Name* ist
- **-group** *gname* findet Dateien die zu der mit *gname* angegebenen Gruppe gehören
- **-size +Nk** findet Dateien die größer als *N* Kilobyte sind
- **-type f, -type d** findet nur reguläre Dateien oder nur Verzeichnisse
- **-mtime +3** findet Dateien die seit 3 Tagen nicht mehr verändert wurden
- **-print** gibt die gefundenen Dateien und Verzeichnisse mir ihrem kompletten Pfad aus
- **-empty** sucht alle leeren Dateien und Verzeichnisse

9.7.2 Aktionen

Gibt man keine Aktion an, so werden die Namen der gefundenen Dateien ausgegeben. Möchte man jedoch Aktionen durchführen so sieht das Format wie folgt aus:

```
find ... -exec Befehl \;
```

Im Befehl wird {} durch den jeweils gefundenen Dateinamen ersetzt.

9.7.3 Beispiele

```
find $HOME -print
```

Listet alle Dateien und Unterverzeichnisse des Homeverzeichnisses auf.

```
find /projekt -name 'menue*' -user student -print
```

Listet alle Dateien des Verzeichnisses */projekt* auf die mit *menue* beginnen und deren Eigentümer *student* ist.

```
find /home -name "*.tar" -user student -size +100k exec rm {} \;
```

Durch diesen Befehl werden im Verzeichnis */home* alle Dateien gefunden die die Endung *.tar* besitzen (Archive), deren Eigentümer der Benutzer *student* ist und die größer als 100 Kilobyte sind. Diese Dateien werden in der Folge ohne Rückfrage gelöscht.

9.8 Weitere Kommandos

Eine kurze Auflistung weiterer nützlicher Kommandos

- **cmp** *Datei1 Datei2* vergleicht die zwei angegebenen Dateien
- **date** gibt Zeit und Datum des Systems aus
- **df** gibt aus wieviel Plattenspeicherplatz belegt und wieviel Speicherplatz frei ist
- **file** *Datei1 ... Datein* ermittelt die Dateitypen der angegebenen Dateien (Dateien durch Leerzeichen getrennt)
- **sort** *Datei* sortiert die Zeilen einer Datei lexikographisch und gibt das Ergebniss aus
- **wc** *Datei* zählt Zeilen, Wörter und Zeichen in der angegebenen Textdatei

- **who** *Datei* gibt die am System angemeldeten Benutzer aus
- **md5sum** *Datei* gibt die md5 Checksumme der Datei aus, dies ist vor allem nützlich um zu überprüfen, ob Dateien die z.B. aus dem Internet heruntergeladen wurden, korrekt übertragen wurden

9.9 Weitere Beispiele

Noch einige Beispiele die das arbeiten mit Pipes und regulären Ausdrücken verdeutlichen.

```
tar cvzf C.tgz $(find . -name "*.c" -mtime -1)
```

Findet und verpackt alle vorhandenen C-Quelltexte des letzten Tages (24 Stunden) verändert wurden in das Archiv *C.tgz*.

```
find . | grep -i abc.txt
```

Sucht nach einer Datei mit dem Namen *ABC.TXT* unabhängig von Groß- oder Kleinschreibung.

```
find / -name xyz.txt -exec less {} \;
```

Finde die Datei *xyz.txt* und zeige diese an.

10 Dienste

In diesem Kapitel sollen einige Dienste vorgestellt werden, welche für Sie vor allen von Interesse sind, wenn Sie beabsichtigen z.B. von Ihrem Internetanschluss zu Hause auf Rechner der BA-Stuttgart zuzugreifen.

Unter Diensten versteht man simpel ausgedrückt Programme, die in der Regel mit dem Hochfahren des Rechners gestartet werden. Sie warten beständig auf Anfragen von Benutzern und beantworten diese.

Ein Beispiel für einen Dienst ist zum Beispiel ein Web-Server. Er wartet beständig auf Anfragen aus dem Internet und beantwortet diese in Form von HTML-Seiten.

Die folgenden Punkte sind der Inhalt dieses Kapitels:

- 10.1 telnet
- 10.2 ftp
- 10.3 ssh
- 10.4 scp

10.1 telnet

Der Dienst *telnet* setzt auf dem gleichnamigen Protokoll Telnet auf. Telnet ist ein Urgestein des Internet. Die endgültige Spezifikation des Telnet Protokols wurde bereits im Jahre 1973 verabschiedet.

Telnet ermöglicht den *textbasierten Zugriff auf entfernte Rechner* (Remote Terminal). Somit können die Resourcen dieses Rechners online genutzt werden. Dies ist ein nicht zu unterschätzender Vorteil, wenn man bedenkt, wie teuer Rechenleistung im Jahre 1973 war. Die Kapazitäten von Großrechnern konnten somit auch von Benutzern genutzt werden die nicht direkt vor Ort waren. Die Eingabe und Ausgabe erfolgt bei einer Telnet-Session ganz normal über die lokalen

Peripheriegeräte. Der Benutzer kann alle Kommandos ausführen als säße er direkt vor der Shell des entfernten Rechners.

Natürlich gibt es auch noch heute komplexe Anwendungen die über ein *Remote Login* auf Hochleistungsrechnern durchgeführt werden. Allerdings hat dieser Anwendungsbereich von Telnet im Laufe der Zeit immer mehr an Bedeutung verloren. Telnet ist ein wunderbares Werkzeug um einen Rechner Remote zu administrieren oder mal eben die Mails vom Universitätsaccount abzufragen.

10.1.1 Kommunikation via Telnet

Telnet ist ein Client-Server Protokoll welches auf TCP basiert. Standardport für Telnet ist *Port 23*.

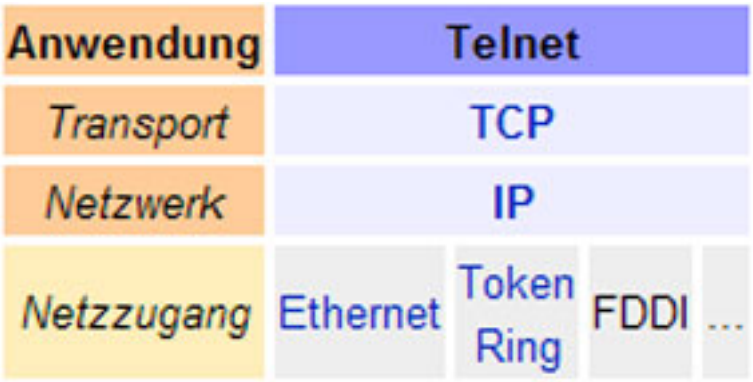

Abbildung 10.1: Telnet im OSI Schichtenmodell [1]

Der Kommunikationsaufbau läuft folgendermaßen ab:

1. **NVT - Network Virtual Terminal**
   Server und Client bilden beim Erstellen der Verbindung ein NVT. Mit Beginn einer Telnet-Session handeln Server und Client mittels NVT alle weiteren Protokoll-Elemente aus. Das NVT bildet somit die Grundstruktur des Telnet-Protokolls. Die Hauptaufgabe des NVT besteht darin unterschiedliche Terminals (bedingt z.B. durch unterschiedliche Betriebssysteme auf Client und Server) miteinander durch Konvertierung kompatibel zu machen. So stellt das NVT beispielsweise an seinen Endpunkten eine virtuelle Tastatur und einen virtuellen Drucker (Monitor) zur Verfügung, welche Eingaben und Ausgaben entsprechend konvertieren.

2. **Negotiated Options**
   Das Aushandeln dieser Optionen garantiert, dass beide Rechner optimal miteinander funktionieren. So kann an dieser Stelle beispielsweise das Tastaturlayout abgeglichen werden.

3. **Symmetrische Verbindung**

In einer Telnet-Verbindung sind beide Rechner gleichberechtigt. Das heisst, dass nicht nur Rechner A kann auf B zugreifen kann, sondern auch umgekehrt Rechner B auf A.

10.1.2 Verwendung des Kommandos telnet

Sie können eine Telnet-Session unter Linux mit dem Befehl

```
telnet
```

oder

```
telnet ferner_Rechner
```

aufbauen. Geben Sie den *fernen_Rechner* mit an so wird automatisch versucht eine Verbindung aufzubauen. Ohne Angabe des fernen Rechners befinden Sie sich in der Telnet Konsole und müssen eine Verbindung mit dem Befehl open unter Angabe des entfernten Rechners öffnen. Der Telnet Server wird Sie zur Eingabe von Login und Passwort auffordern, nach erfolgreichen Login können sie ganz normal in der Konsole arbeiten.

Nützliche telnet Befehle

- **Strg-Z** hält telnet an
- **open** *host [port]* öffnet eine Verbindung zu einem Telnet-Server
- **close** schließt die laufende Telnet-Sitzung und kehrt in den Befehlsmodus zurück
- **status** zeigt den aktuellen Status von telnet an
- **quit** schließt alle offenen Telnet-Sitzungen und beendet das Programm

10.1.3 Nachteile von Telnet

Der Datenverkehr inklusive der Übermittlung von Login und Passwort erfolgt bei Telnet **unverschlüsselt** im Klartext. Das Telnet-Protokoll ist also extrem unsicher. Es empfiehlt sich daher die Verwendung von *ssh* (siehe Kapitel 10.3). *ssh* bietet die selbe Funktionalität wie telnet, der Datenverkehr ist allerdings verschlüsselt.

10.2 ftp

Der *ftp* Dienst basiert auf dem gleichnamigen FTP-Protokoll (File Transfer Protocol). Das FTP-Protokoll ist mit seiner Entstehung im Jahr 1971 das älteste Protokoll des Internet. Mit Hilfe von *ftp* können Sie zwischen zwei auch räumlich weit entfernten Rechnersystemen Daten austauschen. Die Übertragung findet nach dem Client-Server Prinzip statt. Das Betriebssystem der beiden miteinander kommunizierenden Rechner spielt dabei keine Rolle.

10.2.1 Kommunikation via FTP

Zu Beginn jeder FTP Verbindung wird zunächst eine Authentifizierung des Benutzers durchgeführt. Danach erfolgt der Aufbau des *Steuerkanals* über *TCP-Port 21* und des *Datenkanals* über *TCP-Port 20*. Über den Steuerkanal werden ausschließlich FTP-Kommandos übertragen. Die eigentlichen Daten (Dateien) fließen über den Datenkanal.

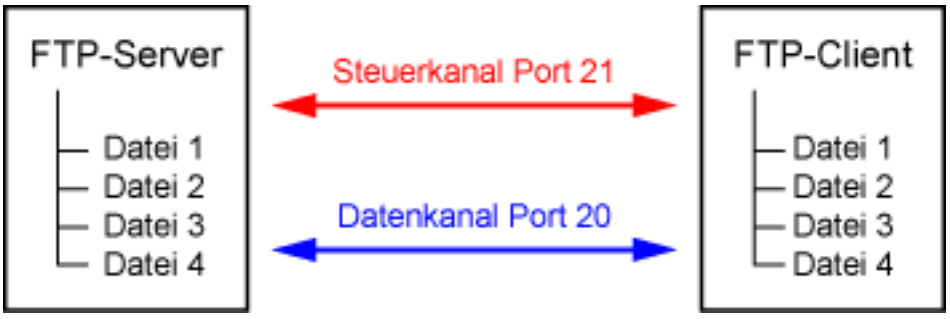

Abbildung 10.2: Schematische Darstellung einer FTP-Verbindung [5]

Nach Abschluss der Datenübertragung wird die Verbindung durch den Benutzer oder nach einem Timeout durch den Server wieder beendet. Die Fehlerkontrolle wird bei der Kommunikation via FTP vollständig dem TCP-Protokoll überlassen.

10.2.2 Verwendung des Kommandos ftp

Im folgenden soll die Verwendung von ftp auf Kommandoebene erläutert werden. Sie können die Verbindung zu einem FTP-Server auf verschiedene Arten aufbauen. Folgende Wege sind möglich:

- `ftp`: Es wird lediglich die ftp-Konsole geöffnet. Die Verbindung zum FTP-Server müssen Sie noch mit dem Befehl *open* herstellen.
- `ftp ferner_Rechner`: Der Rechnername ist bereits angegeben. *ftp* wird versuchen eine Verbindung zu dem Rechner aufzubauen. Benutzer und Passwort werden noch abgefragt.

- `ftp Benutzer@ferner_Rechner:` Eine Verbindung zum *fernen_Rechner* wird aufgebaut und der Benutzer wird bereits übergeben. Das Passwort wird noch abgefragt.

Bewegen in der Dateistruktur

Haben Sie sich am FTP-Server authentifiziert können Sie sich frei, entsprechende Rechte vorausgesetzt, in der Dateistruktur bewegen. Mittels des Befehles

- `dir` können Sie sich den Inhalt eines Verzeichnisses anzeigen lassen. Mit dem Befehl
- `cd` wechseln Sie wie in Linux gewohnt die Verzeichnisse.
- `pwd` zeigt den Namen des aktuellen Arbeitsverzeichniss auf dem entfernten Rechner an.

Dateien kopieren/löschen

Grundsätzlich werden Dateien mit den Befehlen *get* und *put* transportiert.

```
get images/test.gif /myimages/test.gif
```

Mit dem Befehl *get* können Sie eine Datei vom FTP-Server auf Ihren lokalen Rechner übertragen. In diesem Beispiel wird die Datei *test.gif* aus dem Unterverzeichnis *images* auf den lokalen Rechner in das Verzeichnis *myimages* übertragen. Das Verzeichniss *myimages* muss bereits existieren.

```
put /myimages/mytest.gif /images/mytest.gif
```

Mit dem Befehl *put* können Sie Dateien von Ihrem lokalen Rechner auf den FTP-Server übertragen. In diesem Fall wird die lokale Datei *mytest.gif* in das Verzeichnis *images* auf den FTP-Server übertragen.

```
delete /images/mytest.gif
```

Mit dem Befehl *delete* können Sie Dateien auf dem FTP-Server löschen.

Um mehrere Dateien gleichzeitig kopieren zu können stehen die Befehle *mget* und *mput* zur Verfügung. Allerdings führen diese Befehle nur eine Dateinamenserweiterung durch und kopieren letztendlich auch wieder mittels *get* und *put*.

```
mget images/*
```

Überträgt alle Dateien des Verzeichnisses *images* vom FTP-Server ins aktuelle lokale Verzeichnis. Das Unterverzeichnis *images* muss bereits lokal angelegt sein.

`mput /myimages/*`

Überträgt alle Dateien des lokalen Verzeichnisses *myimages* zum FTP-Server.

Weitere nützliche FTP-Befehle

- `bye` beendet die FTP-Verbindung und schließt das Programm (auch *quit* möglich)
- `status` zeigt den aktuellen Status von *ftp* an
- `rmdir` *Verzeichnis* löscht ein Verzeichnis auf dem FTP-Server
- `mkdir` *Verzeichnis* erzeugt ein Verzeichnis auf dem FTP-Server
- `chmod` *[Modus][Datei]* ändert die Zugriffsrechte einer Datei auf dem FTP-Server
- `prompt` schaltet die Abfragen für die Befehle *mget* und *mput* an bzw. aus
- `lcd` *Verzeichnis* wechselt das lokale Arbeistverzeichnis
- `rename` *[Alt][Neu]* benennt eine Datei auf dem FTP-Server um

10.2.3 FTP-Server mit einem Internetbrowser nutzen

Die meisten modernen Internetbrowser bieten die Möglichkeit direkt auf FTP-Server zuzugreifen. Dateien können mit Hilfe dieser Browser um einiges komfortabler bewegt werden.

Der Verbindungsaufbau erfolgt über die Eingabe einer URL der folgenden Form:

`ftp://[Benutzer]:[Passwort]@[Host]`

Host ist dabei die Adresse des FTP-Servers (zB. ftp.suse.de). Bei FTP-Servern die einen anonymen Zugang zulassen, ist die Angabe von Benutzer und Passwort nicht notwendig.

10.2.4 FTP-Clients

Neben der Möglichkeit FTP-Server komfortabel über einen Internetbrowser zu nutzen, existiert eine Vielzahl von FTP-Clients. Dies sind Programme, mit denen FTP komfortabel meisst in der Art eines Dateimanagers genutzt werden kann.

FTP-Clients für Linux

- kbear
- gFTP

FTP-Clients für Windows

- CuteFTP Home
- Total Commander

10.2.5 Anonyme FTP-Server

Es gibt eine große Anzahl von FTP-Servern die auch einen anonymen Zugang zulassen. In diesem Fall ist das Login = `anonymous` und das Passwort die eigene eMail Adresse.

10.3 ssh

ssh (Secure Shell) ist ein Protokoll auf dessen Basis der SSH-Dienst aufsetzt. Standardmäßig erfolgt die Kommunikation via SSH über *Port 22*. Secure Shell bietet vollen Zugriff auf das Dateisystem und alle Funktionen eines entfernten Rechners. Die Kommunikation findet verschlüsselt statt. Aus diesem Grund bietet sich *ssh* als Ersatz für *telnet* an.

Im Augenblick existieren 2 Varianten des SSH-Protokoll, SSH1.x und SSH 2.x. Die beiden Versionen sind miteinander inkompatibel. In der SSH1.x Version sind einige konzeptionelle Schwächen enthalten die einen Angriff auf eine SSH-Verbindung erlauben. Aus diesem Grund sollte stets die neuere Variante des Protokolls verwendet werden.

10.3.1 Kommunikation mit ssh

Die Kommunikation mittels SSH sichert 3 Schwerpunkte ab:

- **Authentifizierung**
  Es wird sichergestellt, dass der Kommunikationspartner auch tatsächlich der ist für, den er sich ausgibt. Hierzu prüft SSH die Identität von Server und Client.

- **Verschlüsselung**
  Alle Daten werden verschlüsselt ausgetauscht. ssh unterstützt z.Zt. die Algorithmen IDEA, DES, 3DES, ARCFOUR und TSS. Standardmäßig wird IDEA benutzt. Beim Verbindungsaufbau handeln die beiden Kommunikationspartner das verwendete Verschlüsselungsverfahren miteinander aus.

- **Integrität**
  Es wird sichergestellt das alle gesendete Daten unverändert beim Empfänger ankommen. Manipulationen werden von SSH erkannt.

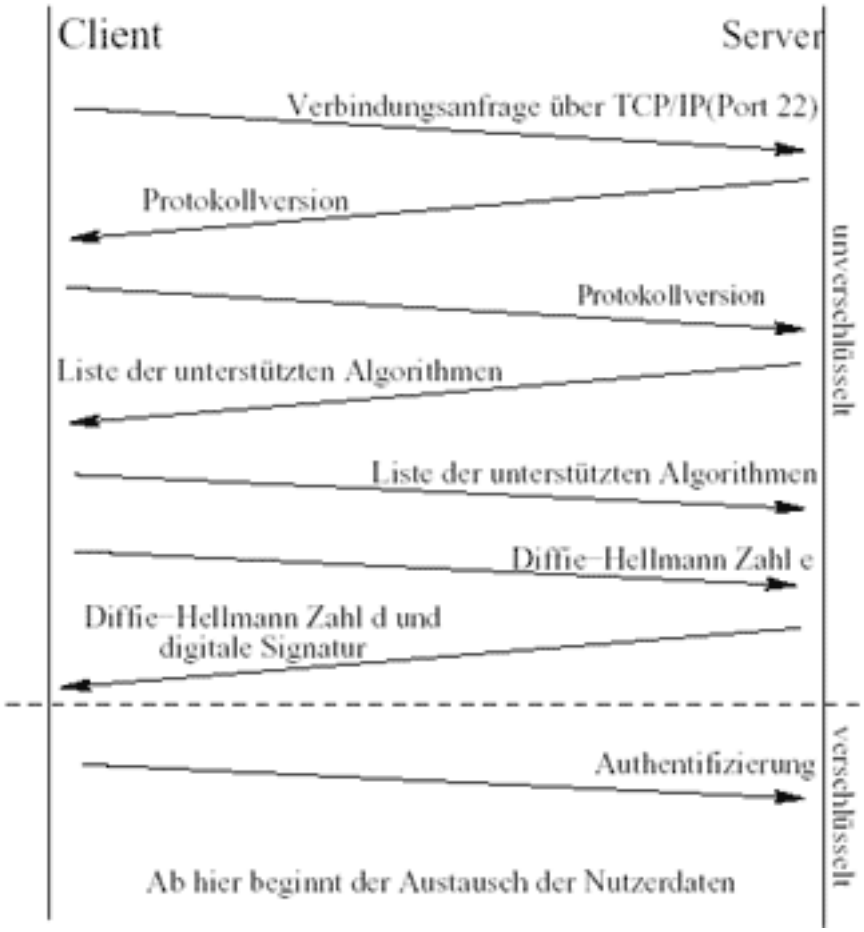

Abbildung 10.3: Schematische Darstellung der Kommunikation [15]

10.3.2 Verbindung zu einem SSH-Server herstellen

SSH2 bietet zwei Möglichkeiten um sich beim Verbindungsaufbau am Server zu authentifizieren. Zum einen *Passwort basiert* und zum anderen *RSA basiert*.

Passwort basierte Authentifizierung

```
ssh login.informatik.ba-stuttgart.de
```

ssh baut mit diesem Befehl eine Verbindung zum BA-Server auf. Bei einem ersten Verbindungsaufbau zu diesem Server werden Sie gefragt ob Sie die Verbindung tatsächlich aufbauen wollen da Ihrem Client der Server noch unbekannt ist. Bestätigen Sie diese Abfrage mit ja (yes) so wird der durch den BA-Server übermittelte Fingerprint bei Ihnen lokal abgelegt. Sie werden nun zur Eingabe ihres Logins und Ihres Passwortes aufgefordert. Diese Daten werden natürlich schon verschlüsselt übertragen. Sinnvoll ist es den Benutzernamen gleich im ssh-Befehl mit anzugeben.

```
ssh benutzer@login.informatik.ba-stuttgart.de
```

Nach erfolgreicher Authentifizierung können Sie ganz normal in der Shell arbeiten. Diese Art der Verbindung weist allerdings einige Schwachpunkte auf. Die RSA basierte Authentifizierung ist hier wesentlich sicherer.

Die SSH-Sitzung können Sie einfach durch Eingabe des Kommandos

```
exit
```

beenden.

RSA basierte Authentifizierung

Um die RSA basierte Authentifizierung benutzen zu können sind zunächst einige kleine Vorarbeiten notwendig um einen *öffentlichen* und einen *privaten Schlüssel* zu erzeugen. Hierzu müssen Sie zunächst mittels

```
ssh-keygen -t rsa
```

einen öffentlichen und einen privaten Schlüssel erzeugen.

Sie werden zunächst nach der Datei gefragt in welcher der Schlüssel gespeichert werden soll. Belassen Sie diese Einstellung wie vorgegeben (Return drücken). Danach werden Sie zur Eingabe einer *Passphrase* aufgefordert. Sie wir zur Verschlüsselung des privaten Schlüssels benötigt. Hier sind Sonderzeichen, Leerzeichen und nummerische Werte erlaubt.

Damit die Authentifizierung per RSA funktioniert müssen Sie jetzt Ihren öffentlichen Schlüssel auf den Server kopieren und in *authorized_keys* umbenennen. Wechseln Sie zunächst in das Verzeichnis in dem der öffentliche Key abgelegt wurde. Benutzen Sie dann folgendes Kommando zum kopieren:

```
rammstein:/ # ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
b3:d3:24:2c:d9:5f:87:11:c8:7c:8b:37:08:a8:a8:2f root@rammstein
rammstein:/ #
```

Abbildung 10.4: Aufruf von ssh-keygen

```
scp id_rsa.pub [user]@[Zielrechner]:/home/[user]/.ssh/authorized_keys
```

Bitte beachten Sie, dass Sie hier Ihren Benutzernamen auf dem entfernten Rechner angeben, zudem muss das Verzeichnis *.ssh* in Ihrem Benutzerverzeichnis auf dem entfernten Rechner bereits existieren. Sie werden zur Eingabe des Passwortes aufgefordert. Nachdem der Kopiervorgang erfolgreich war können Sie sich einfach mittels

```
ssh [user]@entfernter_Rechner
```

am entfernten Rechner anmelden. Nach der Authentifizierung mit der Passpharse können Sie auf dem entfernten Rechner arbeiten.

Die SSH-Sitzung können Sie einfach durch Eingabe des Kommandos

```
exit
```

beenden.

10.3.3 SSH unter Windows

Wenn Sie den ssh Dienst auch unter Windows nutzen möchten, so können Sie dies mit dem Programm Putty tun.

10.4 scp

scp basiert auf *ssh* und kann als Ersatz für FTP verwendet werden, da hier alle Daten verschlüsselt übertragen werden.

Grundsätzlich hat der scp-Befehl die folgende Form:

```
scp [quellRechner]:[quellDatei...] [zielRechner]:[zielDatei...]
```

Wird der Rechner nicht angegeben so wird der lokale Rechner angenommen.

10.4.1 Beispiele

```
scp index.html start.html ferner_Rechner:/usr/index.html
```

Kopiert die Datei *index.html* und *start.html* auf den *fernen_rechner* in das Verzeichnis *usr*. In diesem Fall versucht scp Sie auf dem *fernen_rechner* mit dem selben User anzumelden wie auf Ihrem lokalen System. Sie werden zur Eingabe des Passwortes aufgefordert. Sollten Sie auf dem entfernten Rechner nicht den selben Benutzernamen haben wie auf Ihrem lokalen System so müssen Sie dem Befehl durch ein @ getrennt Ihren Benutzernamen voranstellen:

```
scp index.html [meinBenutzer]@[ferner_Rechner]:/usr/index.html
```

scp lässt auch die Verwendung von regulären Ausdrücken zu:

```
scp -r * [meinBenutzer]@[ferner_Rechner]:/usr/index.html
```

kopiert alle Dateien und Unterverzeichnisse des aktuellen Verzeichnisses.

Um Dateien vom entfernten Rechner auf Ihr System zu übertragen ist ein Befehl in der Form:

```
scp [meinBenutzer]@[ferner_Rechner]:/home/index.html /home/index.html
```

notwenig. In diesem Fall wird die Datei *index.html* an Ihr lokales System übertragen.

10.4.2 Nützliche Paramter

- **-r** kopiert rekursiv alle Unterverzeichnisse mit
- **-q** schaltet die Ausgabe des Kopierfortschrittes aus
- **-p** die Informationen über letzte Änderung, letzten Zugriff und Zugriffsrechte der kopierten Dateien bleiben erhalten
- **-u** löscht die Quelldateien nach dem kopieren
- **-P** Angabe eines alternativen Ports

10.4.3 SCP unter Windows

Auch den scp Dienst können Sie unter Windows benutzen. Beispielsweise mit dem Programm WinSCP.

11 X Window System

Für Linux, wie auch bei anderen Betriebssystemen, gibt es eine grafische Oberfläche für den Benutzer als Teil des sogenannten *X Window System* (vgl. grafische Benutzerschnittstelle). Das folgende Kapitel beschreibt die wichtigsten Elemente und die Konfiguration des X Window Systems am Beispiel von *XFree86*.

Die folgenden Punkte sind der Inhalt dieses Kapitels.

- 11.1 Allgemeines
- 11.2 XFree86; Konfiguration, Display-Manager
- 11.3 X.Org
- 11.4 Window Manager
- 11.5 Desktop-Systeme; KDE, Gnome

11.1 Allgemeines zum X Window System

Das *X Window System* (kurz X) stellt eigentlich eine Sammlung von Basisfunktionen und Protokollen dar, mit deren Hilfe grafische Informationen auf dem Bildschirm ausgegeben und Maus und Tastatur verwaltet werden. Für UNIX und Linux ist das X Window System der Standard für grafische Oberflächen. Die folgende Abbildung zeigt den Aufbau und das Zusammenspiel des X Window Systems mit anderen involvierten Komponenten.

Die einzelnen Komponenten sind wie folgt zu verstehen:

Hardware: Damit sind die notwendigen Komponenten des Rechnersystems gemeint, die es überhaupt ermöglichen ein Bild zu erzeugen und weitergehend eine grafische Oberfläche darzustellen und auch diese zu bedienen: Prozessor, Arbeitsspeicher, Monitor, Grafikkarte und Eingabegeräte wie Tastatur und Maus.

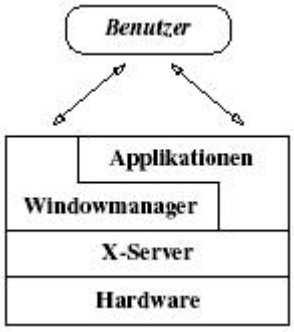

Abbildung 11.1: X Window System, Schichten und Komponenten [20]

X-Server: Der X-Server stellt die Schnittstelle zwischen dem X Window System und der Hardware dar (zum Beispiel: Ansprechen der Grafikkarte), zeichnet grafische Primitive (Punkte, Linien, Rechtecke und Text) und stellt die Netzwerkverfügbarkeit der oberen Schichten bereit.

Window Manager: Der Window-Manager ist ein Programm, welches auf X aufsetzt und für die Verwaltung der Fenster zuständig ist, d.h. er ermöglicht das Wechseln zwischen Fenstern, das Verschieben von Fenstern, das Schließen von Fenstern etc. Wichtig für das Verständnis ist, dass diese Aufgaben vom Window Manager und nicht von X selbst erledigt werden und dass die Funktionsweise daher nicht einheitlich, sondern vom Window Manager abhängig ist.

Applikationen: Die sogenannten X Applikationen bedienen Funktionen, die durch die unterliegenden Komponenten bereitgestellt werden. Ein Beispiel für eine X Applikation ist das Programm xman, welches bereits unter „3. Hilfe unter Linux“ vorgestellt wurde. Die X Applikationen als auch der zuvor angeführte Window Manager sind die Komponenten, mit denen der Benutzer interagiert. X selbst stellt keine Benutzeroberfläche zur Verfügung.

Wie bereits schon beim Gliederungspunkt „X-Server“ angedeutet, ist der X-Server und damit auch die darauf aufsetzenden Komponenten im Netzwerk verfügbar, d.h. man kann eine X Applikation oder den jeweiligen Window Manager auf einem entfernten Rechner ausühren und die Ergebnisse (Fenster etc.) auf dem lokalen Rechner (an dem man arbeitet) darstellen. Zum Vergleich: Eine derartige Funktionalität wird beim Microsoft Windows Betriebssystem erst seit der Version XP Professional mitgeliefert und wurde bei seinen vorangegangenen nur durch die Installation einer zusätzlichen Software für diese Zwecke ermöglicht.

Das X Window System wurde ursprünglich vom Massachusetts Institute of Technology (MIT) entwickelt und besteht gerade in der Version 11 Release 6 (kurz X11R6 oder X11).

11.2 XFree86

XFree86 ist eine freie Implementierung des standardisierten X Window Systems, wird für alle Linux-Distributionen verwendet und steht auch für zahlreiche UNIX-Varianten zur Verfügung. Es ist das Produkt des XFree86 Project, Inc.. XFree86 stand ursprünglich nur für Betriebssysteme mit Intel-Prozessoren zur Verfügung (386, 486, etc., daher die Zahl 86 im Namen), mittlerweile läuft es aber auf sehr vielen Prozessoren. Neben XFree86 gibt es weitere freie Implementierungen wie Xorg und kommerzielle Implementierungen wie Accelerated-X. Die kommerziellen Implementierungen hatten früher eine relativ große Bedeutung, weil XFree86 damals viele moderne Grafikkarten noch nicht unterstützte. Dies hat sich aber gewandelt mit der zunehmenden Leistungsfähigkeit von XFree86. Ganz neue Grafikkarten, die von XFree86 noch nicht unterstützt werden, sind oft auch den kommerziellen Servern unbekannt.

XFree86 Version 3.n und Version 4.n

Aktuelle Distributionen enthalten die Version 4.n (zum Beispiel SuSE 9.1 Professional: 4.3.99.902) von XFree86. Ältere Distributionen enthielten zusätzlich die Version 3.n. Die wichtigsten Vorteile der neueren Version sind:

- Modularisierter X-Server: Der eigentliche Server wird durch ein Modul mit den spezifischen Funktionen für die jeweilige Grafikkarte ergänzt. In der alten Version gab es für jeden Grafikkartentyp einen eigenen Server.
- Unterstützung für TrueType-Fonts
- Anti-Aliasing (Kantenglättung für Fonts)
- Unterstütztung für neue Grafikkarten
- Multi-Head-Betrieb (mehrere Grafikkarten und Monitore gleichzeitig)

Allerdings hat auch die alte Version noch nicht seine Daseinsberechtigung verloren. Einige ältere Grafikkarten werden nur von der alten Version unterstützt. Einige weitere Grafikkarten können zwar mit beiden Versionen betrieben werden, die ältere Version ist aber schneller. Wenn Sie feststellen möchten, welche Version auf Ihrem Rechner verwendet wird, führen Sie den folgenden Befehl auf der Kommandozeile aus:

```
X -showconfig
```

Falls sowohl die Version 3.n als auch 4.n installiert ist, bestimmt die Datei */usr/X11R6/bin/X*, welche tatsächlich verwendet wird. Üblicherweise handelt sich bei X um einen symbolischen Link,

der auf den tatsächlichen Server zeigt. Bei der Version 4.n hat der Server den Namen *XFree86*.

XFree86-Dokumentation

Zu XFree86 existiert eine Menge Online-Dokumentation. Sehr ausführlich sind die Man Pages. Unter anderen existieren Seiten zu:

- **XFree86**: Überblick
- **Xserver**: allgemeine Optionen für den Betrieb von X
- **XF86Config**: Aufbau der Konfigurationsdatei für XFree86

Weitere Informationen zum XFree86 Project und zur gerade aktuellen Version finden Sie im Internet unter `http://www.XFree86.org`.

Starten und Beenden des X-Server

Üblicherweise ist Linux so konfiguriert, dass X beim Systemstart automatisch ausgeführt wird. Für diesen automatischen X-Start ist der Init-V-Runlevel 5 verantwortlich. Wenn der X-Server nicht automatisch gestartet wurde, können Sie ihn manuell über die Kommandozeile mit folgendem Befehl starten:

```
startx
```

Bei erfolgreichem Start sind Sie dann unter X als der Benutzer angemeldet, mit dem Sie den Befehl ausgeführt haben. Der sicherste Weg den X-Server zu stoppen ist der Wechsel in den Runlevel 3 durch Eingabe des folgenden Befehls auf der Kommandozeile:

```
init 3
```

11.2.1 Konfiguration

Die Konfiguration des X-Servers erfolgt normalerweise bei der Installation von Linux. In der Regel läuft X danach zufriedenstellend. Sollten sich jedoch einzelne Komponenten Ihres Rechnersystems nach der Installation ändern (zum Beispiel der Monitor oder die Grafikkarte, müssen Sie eventuell den X-Server entsprechend neu konfigurieren mit Hilfe der Konfigurationsdatei *XF86Config* (wie bereits im vorangegangenen Abschnitt erwähnt). Die Konfiguration von XFree86 erfolgt üblicherweise mit der Datei */etc/X11/XFreeConfig*. Die Konfigurationsdatei darf sich jedoch an

verschiedenen Orten befinden und unterschiedliche Namen haben. Eine vollständige Liste aller Verzeichnisse erhalten Sie auf der entsprechenden Man Page (`man XF86Config-4`).

Konfigurationshilfen

Neben der gängisten Methode zur Konfiguration unter Linux, das Editieren der Konfigurationsdatei mit einem Text-Editor (vgl. Editor vi), werden je nach Distribution zusätzliche Werkzeuge zum Konfigurieren bereitgestellt, die bei der Konfiguration mehr oder weniger Hilfestellung geben:

- **XFdrake:** Mandrake Linux
- **Xconfigurator:** Red Hat Linux
- **SAX2:** SuSE Linux
- **x86cfg / xf86config:** Diese Konfigurationsprogramme sind Teil des X-Basissystems und sollte daher unabhängig von der Distribution zur Verügung stehen.

Struktur der XF86Config

Die Konfigurationsdatei in mehrere Abschnitte gegliedert, die mit Section „name“ beginnen und `EndSection` abgeschlossen werden. Die wichtigsten Abschnitte sind wie folgt:

- **ServerLayout:** verbindet die Abschnitte „Screen“ und „InputDevices“. Mehrere „ServerLayout“-Abschnitte können beim Betrieb von mehreren Grafikkarten und Monitoren benutzt werden.
- **Files:** Dateinamen. Dieser Abschnitt ist dafür vorgesehen, Dateipfade zu Fonts und Farbinformationen hinzuzufügen.
- **InputDevice:** Mehrere Abschnitte diesen Typs werden verwendet, um mindestens die Tastaur und die Maus einzubinden.
- **Monitor:** In diesem Abschnitt werden die Leistungsmerkmale des benutzten Monitors spezifiziert.
- **Device:** In diesem Abschnitt wird die Grafikkarte und das entsprechende Modul für den X-Server spezifiziert.
- **Screen:** Dieser Abschnitt verknüpft „Monitor“- und „Device“-Abschnitte. In den Unterabschnitten werden die erlaubten Bildschirmauflösungen definiert. Mehrere „Screen“-Abschnitte können für die Verwaltung mehrerer „Monitor“/ „Device“-Paare verwendet werden.

Abbildung 11.2 zeigt beispielhaft den Inhalt der XF86Config (Zur besseren Darstellung sind nur die wesentlichen Inhalte enthalten).

```
Section "ServerLayout"
  Identifier    "Layout[all]"
  InputDevice   "Keyboard[0]" "CoreKeyboard"
  InputDevice   "Mouse[1]" "CorePointer"
  Screen        "Screen[0]"
EndSection

Section "Files"
EndSection

Section "InputDevice"
  Driver        "Keyboard"
  Identifier    "Keyboard[0]"
EndSection

Section "InputDevice"
  Driver        "mouse"
  Identifier    "Mouse[1]"
  Option        "Device" "/dev/mouse"
  Option        "Protocol" "imps/2"
EndSection

Section "Monitor"
  HorizSync     31-68
  Identifier    "Monitor[0]"
  ModelName     "1024X768@85HZ"
  VendorName    "--> VESA"
  VertRefresh   50-85
EndSection

Section "Device"
  BoardName     "GeForce2 MX/MX 400"
  BusID         "1:0:0"
  Driver        "nv"
  Identifier    "Device[0]"
  VendorName    "NVidia"
EndSection

Section "Screen"
  DefaultDepth 24
  SubSection "Display"
    Depth       24
    Modes       "1024x768"
  EndSubSection
  Device        "Device[0]"
  Identifier    "Screen[0]"
  Monitor       "Monitor[0]"
EndSection
```

Abbildung 11.2: Beispiel XF86Config

Für Anfänger empfiehlt es sich eines der oben genannten Werkzeuge, soweit verfügbar, zur Konfiguration zu verwenden oder bei speziellen Problemen nach HOWTOs im Internet zu schauen.

11.2.2 Display-Manager

Wie bereits am Anfang dieses Abschnitts erwähnt, wird X bei fehlerfreier Funktion im Rahmen des Init-V-Prozesses im Runlevel 5 gestartet. Dabei wird X durch einen Display-Manager gestartet. Eine weitere Aufgabe des Display-Manager ist das Benutzer-Login. Sichtbar wird dies für den Benutzer nach Abschluss des Init-V-Prozesses, wenn eine grafische Login-Box erscheint. Mit dieser muss sich der Benutzer mit seinem Benutzernamen und seinem Passwort authentifizieren. Ist der Login erfolgreich, wird der entsprechende Window Manager oder das entsprechende Desktop-System gestartet (später mehr dazu). Dieser Benutzer-Login kann als grafischer Ersatz für die Login-Eingabeaufforderung, wie man sie von der Kommandozeile kennt, gesehen werden. Darüber hinaus ist der Display-Manager in der Lage entfernte X-Server zu verwalten, kann also auch Benutzer-Logins auf entfernten X-Servern bereitstellen. Unter Linux gibt nicht nur einen Display-Manager, sondern eine ganze Reihe:

- **xdm:** Der X Display-Manager ist zu X zugehörig und schon am längsten verfügbar.
- **kdm:** Der KDE Display-Manager ist ein Teilprodukt des KDE Desktop-Projektes und weitestgehend kompatibel zum xdm.
- **gdm:** Der Gnome Display-Manager ist ein Teilprodukt des Gnome Desktop-Projektes und ebenfalls weitestgehend kompatibel zu *xdm*.

Obwohl alle drei Display-Manager weitestgehend kompatibel zueinander sind, besitzen alle mehr oder weniger unterschiedliche Zusatzfunktionen gegenüber den anderen. Des weiteren unterscheiden sich alle sowohl in ihrer Konfiguration, als auch in der Struktur ihrer Konfigurationsdateien. Daher sollte Ihnen an dieser Stelle ein erster Überblick über die verschiedenen Display-Manager und deren generelle Funktionen beziehungsweise Aufgaben geboten werden. Weitere Informationen zu den einzelnen Display-Managern finden unter folgenden Links:

- The X Display Manager
- The kdm handbook
- Gnome Display Manager Reference Manual

11.3 X.Org

X.Org ist ein alternativer X-Server. Er basiert auf dem Quellcode von XFree86 Version 4.3.99. Dies war die letzte XFree86 Version, der von den Programmieren noch nicht die Bedingung auferlegt

wurde, dass Software für das X Window System von nun an eine Art Danksagung an die Programmierer von XFree86 enthalten musste. Nach Ansicht der X.Org Foundation [18] läuft dieses Gebahren dem Gedanken von OpenSource-Software zuwider. Aufgrund der Code-Verwandtschaft lassen sich Konfigurationsdateien sowie Anwendungen (Windowmanager, Desktopsysteme, etc.) weitgehend ohne Anpassung aus einer XFree86-Umgebung übernehmen.

11.4 Window Manager

Wie bereits am Anfang dieses Kapitels erwähnt wurde, stellen Window Manager eine grundlegende grafische Oberfläche dar. Sie betten X Applikationen in ihre entsprechenden Fenster ein, um diese dann für den Benutzer bedienbar zu machen (zum Beispiel das Verschieben oder das Schließen eines Fensters). Zusätzlich bieten sie meist ein kompaktes Menü, um X Applikationen direkt und nicht von der Kommandozeile starten zu können. Generell gilt für alle Window Manager, dass sie sehr kompakt und leistungsstark sind, da sie auf die „wesentlichen Dinge“ reduziert sind. Auf der anderen Seite sind sie für Anfänger sehr ungewohnt, da ihre Bedienungsweise keine Ähnlichkeit zu grafischen Benutzerschnittstellen anderer Betriebssysteme hat (vgl. Microsoft Windows, MacOS, etc.).

Beispielhaft soll hier der Window Maker gezeigt werden, der bei der Standardinstallation der meisten Distributionen installiert wird (siehe Abbildung 11.3).

Die Konfigurationsprogramme für den Window Maker und die gerade ausgeführten X Applikationen werden durch „Kacheln“ am Rand der Oberfläche repräsentiert. Oben rechts in der Abbildung sehen wir das quaderförmige Menü des Window Maker, welches mit einen Klick auf die rechte Maustaste ausgewählt werden kann.

Eine gute Übersicht aktuell verfügbarer Window Manager finden Sie unter `http://xwinman.org/`.

11.5 Desktop-Systeme

Desktop-Systeme gehen über die Funktion eines Window Managers noch weit hinaus. Aufgrund von grafischen Bibliotheken sehen viele Anwendungsprogramme einheitlich aus und sind auch einheitlich zu bedienen. Die Kommunikation zwischen den Programmen ist standardisiert, dass beispielsweise Drag-und-Drop-Kommandos zwischen unterschiedlichen Programmen möglich sind etc. Im Folgenden sind einige Merkmale aufgezeigt:

- Ein Desktop-System bietet eine **Benutzeroberfläche** für elementare Operationen (Umgang

Abbildung 11.3: Window Maker

mit Dateien etc.).

- Die Bedienung ist so **intuitiv** wie möglich (Drag&Drop, Icon-Symbole), besitzt ein hohes Maß an Benutzerfreundlichkeit.
- Bereitstellung einer Sammlung von grundlegenden **einheitlich zu bedienenden Anwendungsprogrammen** für häufig benötigte Operationen (zum Beispiel ein Text-Editor oder ein Taschenrechner).
- Alle Desktop-Komponenten sind **einfach zu konfigurieren** (durch Dialoge, nicht durch kryptische Textdateien; vgl. Benutzerfreundlichkeit).
- Zu allen Komponenten beziehungsweise Programmen steht eine **umfassende Online-Dokumentation** in einem leicht zu lesenden Format (HTML) zur Verfügung.
- Der **Austausch und die Nutzung von Daten** unterschiedlicher Desktop-Programme sind unkompliziert.

- **Internet-Funktionen** (etwa der Zugriff auf ftp-Verzeichnisse) werden von allen Komponenten transparent unterstützt. Lokale und externe Daten können auf die gleiche Weise bearbeitet werden.

Im Folgenden werden die beiden Deskop-Systeme, KDE und Gnome, vorgestellt, die die meiste Popularität genießen und daher mit den meisten Distributionen mitgeliefert werden.

11.5.1 KDE (The K Desktop Environment)

KDE ist ein zeitgemäßes Desktop-System mit Netzwerktransparenz für UNIX und Linux. KDE versucht dem Bedarf eines einfach zu bediendenden Desktops für UNIX und Linux nachzukommen, ähnlich den Desktops wie man sie bei Microsoft Windows oder MacOS findet. Das KDE Project wurde im Oktober 1996 gegründet und steht unter der LGPL / GPL Lizenz. Die folgende Abbildung zeigt beispielhaft KDE 3.1, wie es bei SuSE Linux 9.0 Professional mitgeliefert wird, beispielhaft deswegen, weil KDE eine Vielzahl von Anpassungsmöglichkeiten bietet, so dass es kein Standarderscheinungsbild für KDE gibt.

Grundsätze für KDE

In KDE wird standardmäßig alles, außer in Ausnahmefällen, mit einem einfachen Mausklick gestartet (kann aber auch entsprechend mit dem KDE-Kontrollzentrum umkonfiguriert werden).

- Das Kontextmenü kann bei vielen Programmen beziehungsweise KDE-Komponenten mit der rechten Maustaste aufgerufen werden.

- Fast alle KDE-Programme kommen mit Drag&Drop zurecht. Sind weitere Angaben durch den Benutzer notwendig, öffnet sich ein Kontextmenü und erfragt diese.

- KDE selbst wird über das bereits im ersten Gliederungspunkt genannte KDE-Kontrollzentrum konfiguriert. Zur Konfiguration einzelner KDE-Programme stehen entsprechende Menü-Kommandos zur Verfügung.

- Verlassen Sie KDE stets über den entsprechenden Menüpunkt im KDE-Start-Menü (Abbildung 11.4, unten links). KDE speichert nämlich Informationen über aktuell laufende Programme und versucht diesen Stand möglichst beim nächsten Start von KDE wiederherzustellen.

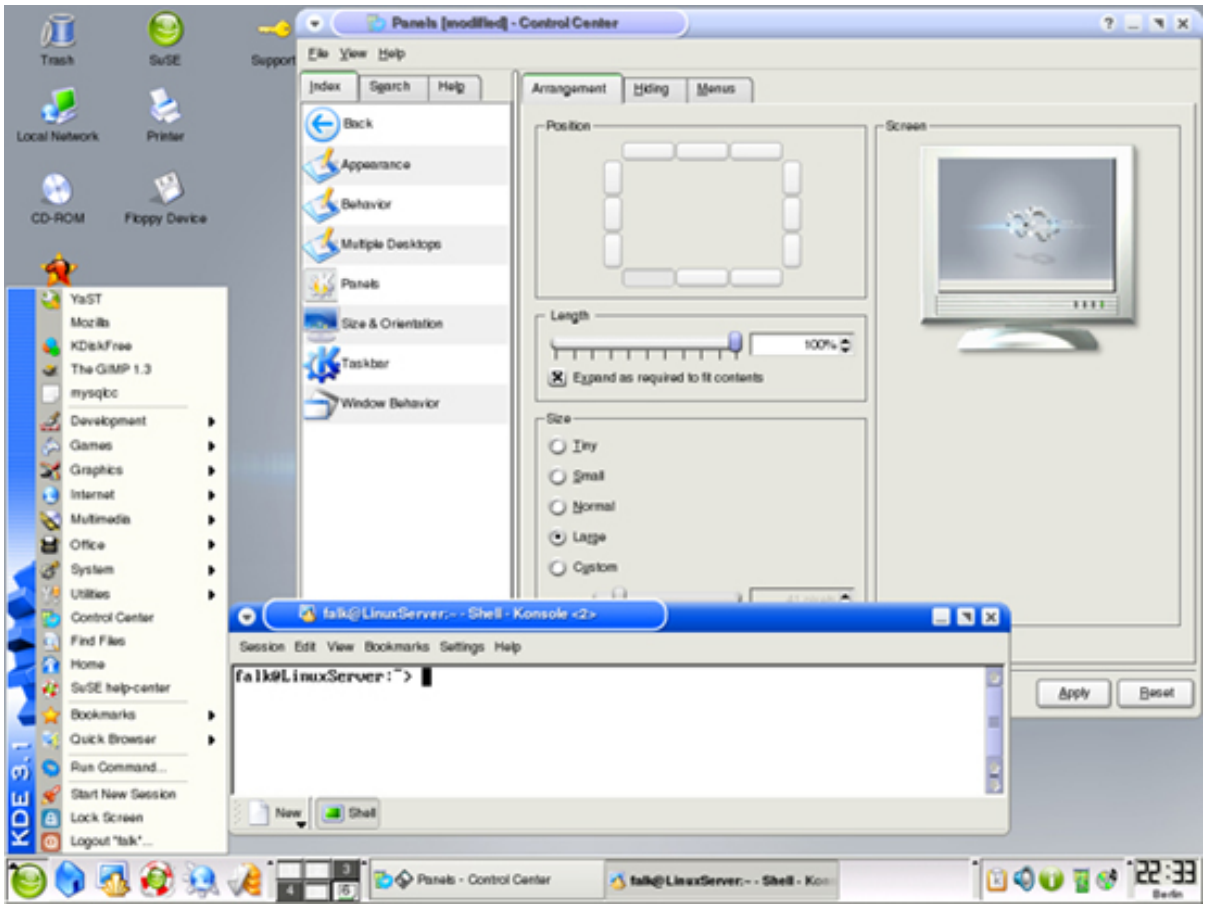

Abbildung 11.4: KDE 3.1

Der Desktop im Überblick

An dieser Stelle soll Ihnen ein Überblick über die wesentlichen Elemente des KDE Desktop gegeben werden (vgl. Abbildung 11.4):

Icons: Der Desktop kann mit beliebigen Icons gefüllt werden (siehe Abbildung 11.4, oben links), die einen schnellen Zugriff auf diverse Programme und Geräte erlauben. Standardmäßig sind bereits Icons zum Beispiel für Wechsellaufwerke (CD-ROM,DVD-ROM, etc.) oder Open Office (Office-Anwendung) , falls installiert, vorhanden.

Panel: Das Panel (siehe unterer Rand in Abbildung 11.4) spielt eine zentrale Rolle bei der Bedienung von KDE und kann in seinen Funktionen mit der Task-Leiste, wie man sie von Microsoft Windows kennt, verglichen werden. Das KDE-Startmenü, über das man die installierten (KDE-)Programme aufrufen kann, wird über das erste Symbol links im Panel (meist das Symbol des KDE Project, hier das SuSE Symbol) aufgerufen. Danach folgen Symbole (von links nach rechts) für das eigene Home-Verzeichnis, die Kommandozeile, das

KDE Help Center, den Konqueror (Dateimanager & Browser) und KMail (KDE eigener Email Client). Danach folgt der Pager (Umschalter), der es ermöglicht zwischen mehreren virtuellen Desktops zu wechseln (standardmäßig vier). In der Mitte findet sich die Taskbar (Fensterleiste), in der die laufenden Programme angezeigt werden. Auf der rechten Seite des Panel finden sich Symbole für KDE eigene Programme, die ständig im Hintergrund bereitstehen (Abbildung 11.4 Klipper, KAMix, SuSEPlugger und kscd) und eine Uhr.

Konsole: Konsole (Abbildung 11.4, Mitte unten) ist ein Kommandozeilen-Emulator für KDE. Er besitzt den großen Vorteil mehrere Kommandozeilen parallel verwalten zu können.

KDE-Kontrollzentrum: Wichtige KDE-Komponenten werden über das KDE-Kontrollzentrum (siehe Abbildung 11.4, oben rechts) konfiguriert. Der Befehl dafür auf der Kommandozeile lautet „kcontrol“ Das Kontrollzentrum öffnet den Zugang zu unzähligen Einzeldialogen, jedoch bedürfen die Konfigurationsmöglichkeiten keiner weiteren Erläuterung.

Wichtige KDE Werkzeuge

An dieser Stelle sollen noch einige wichtige KDE-Programme genannt werden, die bis jetzt noch keine Erwähnung gefunden haben.

kedit: Hierbei handelt es sich um einfachen Text-Editor.

kwrite/ kate: kwrite bietet den Vorteil gegenüber kedit, dass er für eine Reihe etablierter Programmiersprachen eine automatische Syntaxhervorhebung bietet. Dies macht besonders für Programmierer interessant. Seit der Version 2.2 steht darüber hinaus das Programm kate zur Verfügung, dass noch mehr Konfigurationsmöglichkeiten bietet.

kile: Komfortabler Editor für LaTeX-Dateien

kghostview: Dieses Programm dient zum Betrachten von PDF- und PostScript-Dateien (vgl. *ghostview*).

kdvi: Dieses Programm ermöglicht das komfortable Lesen von DVI-Dateien (vgl. *xdvi*).

kview: Mit diesem Programm können Sie diverse Bitmap-Dateien betrachten.

ksnapshot: Mit diesem Programm können Sie Screenshots erstellen, wie sie auch in diesem Kurs zu finden sind.

kdf: Dieses Programm zeigt für alle Datenträger grafisch ansprechend aufbereitet an, wie viel Speicher noch frei ist (siehe */etc/fstab*).

kpm/ kysguard: Diese Programme zeigen eine Liste aller laufenden Prozesse (vgl. *top*), wobei ksysguard deutlich mehr Funktionen zur detaillierten Systemanalyse bietet.

11.5.2 Gnome

Das Gnome Project (GNU Network Object Model Environment) soll einen intuitiven und ansprechenden Desktop für den Benutzer bieten. Gnome ist Teil des GNU Project. Der Hauptunterschied zwischen KDE und Gnome, wovon stets auch der Konflikt, welches nun das bessere Desktop-System ist, ausgeht, sind die Bibliotheken, auf den diese beiden Desktop-Systeme basieren. KDE basiert auf Qt, einer kommerziellen Bibliothek der Firma Trolltech. Diese darf zwar für freie Softwareprojekte kostenlos genutzt und weitergegeben werden. Es galten jedoch für die Version 1.n eine Menge Einschränkungen, die sich aber gelockert haben aufgrund der neuen QPL (Q Public Licence). Diese ist zwar nicht so „frei“ wie die LGPL, aber doch frei genug, um von der Open-Source-Entwicklungsgemeinde akzeptiert zu werden.

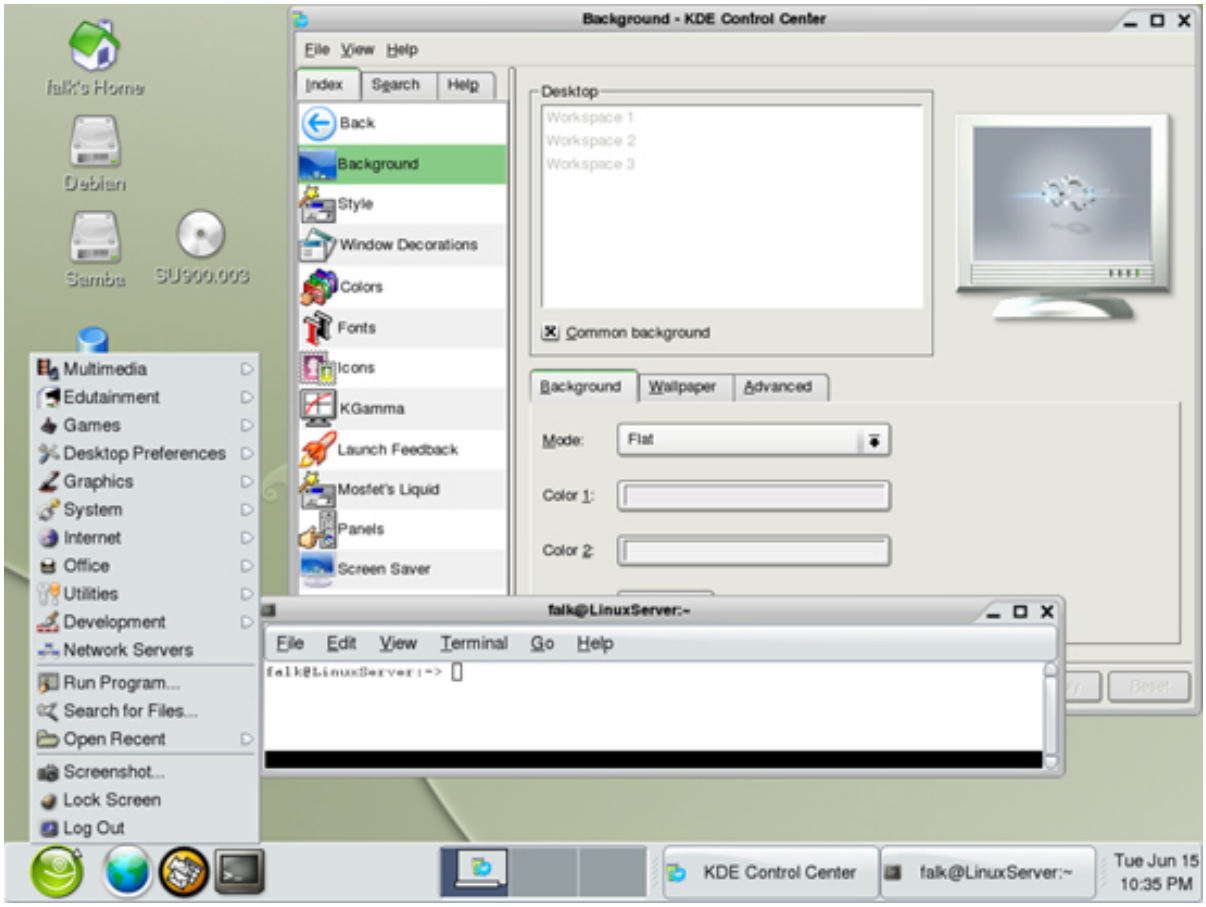

Abbildung 11.5: Gnome 2

Gnome basiert dagegen auf dem GTK (Gimp Tool Kit), also einer Bibliothek, die ursprünglich nur für die Benutzeroberfläche von Gimp entwickelt wurde (Gimp ist ein Bildverarbeitungsprogramm). Die Gtk ist wirklich freie Software und untersteht der LGPL. Abbildung 11.5 zeigt beispielhaft Gnome 2, wie es bei SuSE Linux 9.0 Professional mitgeliefert wird, beispielhaft deswegen, weil auch

Gnome eine Vielzahl von Anpassungsmöglichkeiten bietet, so dass es kein Standarderscheinungsbild für Gnome gibt.

Auf den ersten Blick erkennt man die Ähnlichkeit zwischen KDE und Gnome. Eine Zusammenführung dieser beiden Desktop-Systeme wäre eine sinnvolle Lösung nach Änderung der rechtlichen Lage, jedoch kam diese viel zu spät.
Gnome bietet eine nicht so breite Palette an eigenen Programmen wie man es bei KDE gewohnt ist. Auf anderen Seite muss man aber auch beachten, dass viele KDE-Programme Gegenstücke (mit erweiterter Funktionalität) zu bereits vorhandenen Open Source Programmen sind.

Grundsätze für Gnome

- Im Gegensatz zu KDE ist in Gnome zum Auslösen von Icon-Aktionen (also etwa zum Start eines Programms) meistens ein Doppelklick notwendig. Eine Ausnahme stellen allerdings Icons im Panel dar: Bei diesen Icons reicht ein einzelner Klick aus.
- Bei sehr vielen Programmen beziehungsweise Gnome-Komponenten kann man mit der rechten Maustaste ein Kontextmenü aufgerufen werden.
- Fast alle Gnome-Programme kommen mit Drag&Drop zurecht. Wenn Sie zusätzlich zur linken Maustaste die Strg-Taste drücken, kopieren gleich das Objekt.
- Gnome sollten man ebenfalls über den entsprechenden Menüpunkt des Startmenüs, da Gnome ebenfalls Informationen über die gerade laufenden Programme speichert.

A Literaturverzeichnis

[1] *das ELKO*. http://www.elektronik-kompendium.de.

[2] *Free Software Foundation*. http://www.fsfeurope.org/.

[3] *Gnome*. http://www.gnome.org.

[4] *GNU Project*. http://www.gnu.org.

[5] *GTK - The GIMP Toolkit*. http://www.gtk.org.

[6] *IEEE - POSIX Cerfification Authority*. http://standards.ieee.org/regauth/posix/.

[7] *KDE*. http://www.kde.org.

[8] *Linux Documentation Project*. http://www.linuxdoc.org.

[9] *Linux Information Systems AG*. http://www.linux-ag.de/linux/LHB/node153.html.

[10] *Linux Standard Base*. http://www.linuxbase.org.

[11] *Pathname - Filesystem Hierarchy Standard*. http://www.pathname.com/fhs/.

[12] *Trolltech - Q Public License*. http://www.trolltech.com/licenses/qpl-annotated.html.

[13] *UnitedLinux*. http://www.unitedlinux.com/.

[14] *Vim - the editor*. http://www.vim.org.

[15] *Wikipedia - Die freie Enzyklopädie*. http://de.wikipedia.org.

[16] *Window Maker*. http://windowmaker.org.

[17] *The XFree86 Project,Inc.* http://www.xfree86.org.

[18] *X.Org*. http://www.x.org.

[19] Dean, Jeffrey: *LPI Linux Certification in a Nutshell*. O'Reilly Verlag, 1st edition, May 2001.

[20] Herrler, Hans-Jürgen: *Praktische Datenverarbeitung - Einführung in Linux / Unix*, 6. Auflage.

[21] Kofler, Michael: *Linux - Installation, Konfiguration und Anwendung*. Addison-Wesley, 6. Auflage, 2001. An imprint of Pearson Education.

[22] Stroetmann, Karl: *TCL - Skript*. Vorlesungsskript, Berufsakademie Stuttgart, 2003.

[23] Weber, Roland: *Betriebssysteme*. Vorlesungsskript, Berufsakademie Stuttgart, IBM Deutschland Entwicklung, 2004.